高职高专计算机任务驱动模式教材

ASP.NET动态网站开发项目化教程

程琪 张白桦 编著



高职高专计算机任务驱动模式教材

ASP.NET 动态网站开发项目化教程

程 琪 张白桦 编著

清华大学出版社

北 京

内 容 简 介

本书以项目化任务为载体，系统介绍了微软 ASP.NET 动态网站开发技术。全书共分 10 章，主要内容包括 ASP.NET 框架与环境、C#编程、ASP.NET 服务器控件和第三方控件、ASP.NET 系统对象、数据库访问和编程、XML 访问、Web 服务、网站部署与安全性配置、AJAX 技术等。

本书以一个 MVC 三层架构的实际的门户网站项目为主线，突出.NET 的特点和应用方向，强调“边做边学”，将理论知识分解到一个项目的众多任务中去，并给出了每个任务实例的设计步骤和源代码。

0931 项目包括一个多功能多角色的三层架构新闻网站、一个同样多功能的三层架构博客网站、一个基于系统对象设计的聊天室、一个基于 XML 技术的留言板、一个提供电话区号查询的 Web 服务、一个基于 AJAX 技术的新闻展示改进方案以及网站后台管理、网站配置和安全性策略等。

本书结构清晰、浅显易懂，既可作为高职高专院校计算机专业 Web 开发课程教材以及相关培训教材，也适合具有一般计算机基础的读者自学使用。

图书在版编目(CIP)数据

ASP.NET 动态网站开发项目化教程/程琪，张白桦编著. —北京：清华大学出版社，2010.1
高职高专计算机任务驱动模式教材
ISBN 978-7-302-21737-4

Ⅰ. A…　Ⅱ. ①程…　②张…　Ⅲ. 主页制作—程序设计—高等学校：技术学校—教材
Ⅳ. TP393.092

中国版本图书馆 CIP 数据核字(2009)第 243790 号

责任编辑：张　景
责任校对：袁　芳
责任印制：

出版发行：清华大学出版社　　**地　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销：全国新华书店
开　本：185×260　　**印　张**：15.25　　**字　数**：348 千字
版　次：2010 年 1 月第 1 版　　**印　次**：2010 年 1 月第1 次印刷
印　数：1～　000
定　价：　.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。联系电话：010-62770177 转 3103　　产品编号：033655-01

丛书编委会

出版说明

我国高职高专教育经过近十年的发展，已经进入深度教学改革阶段。教育部2006年12月发布了教高[2006]16号文件“关于全面提高高等职业教育教学质量的若干意见”，文件指出，应大力推行工学结合，突出实践能力培养，全面提高高职高专教学质量。

清华大学出版社作为国内大学出版社的领跑者，为了进一步推动高职高专计算机专业教材的建设工作，适应高职高专院校计算机类人才培养的发展趋势，根据教高[2006]16号文件的精神，2007年秋季开始了切合新一轮教学改革的教材建设工作。

目前国内高职高专院校计算机网络与软件专业的教材品种繁多，但切合国家计算机网络与软件技术专业领域技能型紧缺人才培养培训方案并符合企业的实际需要、能够成体系的教材还不成熟。

我们组织国内对计算机网络和软件人才培养模式有研究并且有实践经验的高职高专院校，进行了较长时间的研讨和调研，遴选出一批富有工程实践经验和教学经验的双师型教师，合力编写了这套适用于高职高专计算机网络、软件专业的教材。

本套教材的编写方法是以任务驱动案例教学为核心，以项目开发为主线。我们研究分析了国内外先进职业教育的培训模式、教学方法和教材特色，消化吸收优秀的经验和成果。以培养技术应用型人才为目标，以企业对人才的需要为依据，把软件工程和项目管理的思想完全融入教材体系，将基本技能培养和主流技术相结合。课程设置中重点突出、主辅分明、结构合理、衔接紧凑。教材侧重培养学生的实际操作能力，学、思、练相结合，旨在通过项目实践，增强学生的职业能力，使知识从书本中释放并转化为专业技能。

一、教材编写思想

本套教材围绕开发项目所用到的知识点进行讲解，对某些知识点附上相关的例题，以帮助读者理解，进而将知识转变为技能。

考虑到是以“项目设计”为核心组织教学，所以在每一学期都配有相应的实训课程及项目开发手册，要求学生在教师的指导下，能整合本学期所学的知识内容，相互协作，综合应用该学期的知识进行项目开发。同时在

教材中采用了大量的案例，这些案例紧密地结合教材中的各个知识点，循序渐进，由浅入深，在整体上体现了内容主导、实例解析，以点带面的模式，配合课程后期以项目设计贯穿教学内容的教学模式。

软件开发技术具有种类繁多、更新速度快的特点。本套教材在介绍软件开发主流技术的同时，帮助学生建立软件相关技术的横向及纵向的关系，培养学生综合应用所学知识的能力。

二、丛书特色

本系列教材体现目前的工学结合教改思想，充分结合教改现状，突出项目面向教学和任务驱动模式教学改革成果，打造立体化精品教材。

(1) 参照或吸纳国内外优秀计算机网络、软件专业教材的编写思想，采用本土化的实际项目或者任务，以保证其有更强的实用性，并与理论内容有很强的关联性。

(2) 准确把握高职高专软件专业人才的培养目标和特点。

(3) 充分调查研究国内软件企业，确定了基于 Java 和.NET 的两个主流技术路线，再将其组合成相应的课程链。

(4) 教材通过一个个的教学任务或者教学项目，在做中学，在学中做，以及边学边做，重点突出技能培养。在突出技能培养的同时，还介绍解决思路和方法，培养学生未来在就业岗位上的终身学习能力。

(5) 借鉴或采用项目驱动的教学方法和考核制度，突出计算机网络、软件人才培训的先进性、工具性、实践性和应用性。

(6) 以案例为中心，以能力培养为目标，并以实际工作的例子引入概念，符合学生的认知规律。语言简洁明了、清晰易懂、更具人性化。

(7) 符合国家计算机网络、软件人才的培养目标；采用引入知识点、讲述知识点、强化知识点、应用知识点、综合知识点的模式，由浅入深地展开对技术内容的讲述。

(8) 为了便于教师授课和学生学习，清华大学出版社正在建设本套教材的教学服务资源。在清华大学出版社网站（www.tup.com.cn）免费提供教材的电子课件、案例库等资源。

高职高专教育正处于新一轮教学深度改革时期，从专业设置、课程体系建设到教材建设，依然是新课题。希望各高职高专院校在教学实践中积极提出意见和建议，并及时反馈给我们。清华大学出版社将对已出版的教材不断地修订、完善，提高教材质量，完善教材服务体系，为我国的高职高专教育继续出版优秀的高质量的教材。

清华大学出版社
高职高专计算机任务驱动模式教材编审委员会
rawstone@126.com

序

教材是根据课程标准而编写的，而课程又是根据专业培养方案而设置的，高职专业培养方案是以就业为导向，基于职业岗位工作需求而制订的。在高职专业培养方案的制订过程中，必须遵照教育部教高[2006]16 号文件的精神，体现工学结合人才培养模式，重视学生校内学习与实际工作的一致性。制订课程标准，高等职业院校要与行业企业合作开发课程，根据技术领域和职业岗位(群)的任职要求，参照相关的职业资格标准，改革课程体系和教学内容。在教材建设方面，应紧密结合行业企业生产实际，与行业企业共同开发融“教、学、做”为一体，强化学生能力培养的实训教材。

教材既是教师教的资料，又是学生学的资料。在教学过程中，教师与学生围绕教材的内容进行教与学。因此，要提高教学质量必须有一套好的教材，赋之于教学实施。

高等职业技术教育在我国仅有 10 年的历史，在专业培养方案制订、课程标准编制、教材编写等方面还都处于探索期。目前，高职教育一定要在两个方面下工夫：一是职业素质的培养；二是专业技术的培养。传统的教材，只是较为系统地传授专业理论知识与专业技能，大多数是从抽象到抽象，这种教学方式高职院校的学生很难接受，因为高职学生具备的理论基础与逻辑思维能力远不及本科院校的学生，因此传统体系的教材不适合高职学生的教学。

认识的发展过程是从感性认识到理性认识，再由理性认识到能动地改造客观世界的辩证过程。一个正确的认识，往往需要经过物质与精神、实践与认识之间的多次反复。“看图识字”、“素描临摹”、“师傅带徒弟”、“工学结合”都是很好的学习模式，因此以案例、任务、项目驱动模式编写的教材会比较适合高职学生的学习，让学生从具体认识到抽象理解，边做边学，体现“做中学、学中做”，不断循环，从而完成职业素养与专业知识和技能的学习，尤其在技能训练方面得到加强。学生在完成案例、任务、项目的操作工作中，掌握了职业岗位的工作过程与专业技能，在此基础上，教师用具体的实例去讲解抽象的理论，显然是迎刃而解。

清华大学出版社与杭州开元书局共同策划的“高职高专计算机任务驱动模式教材”，就是遵照教育部教高[2006]16 号文件精神，综合目前高

职院校信息类专业的培养方案和课程标准,组织有多年教学经验的一线教师进行编写。教材以案例、任务、项目为驱动模式,结合最前沿的IT技术,体现职业素养与专业技术。同时,充分考虑教学目标、教师、学生、实训条件,从而使教材的结构与内容适合教师能教、学生能学、实训条件能满足,真正成为高等职业技术教育的合理化教材,以推动高职教材改革和创新的发展。

在教学实施过程中,以案例、任务、项目为驱动已经得到教师与学生的认可,但用教材进行充分体现尚属于尝试阶段。清华大学出版社与开元书局在这方面进行大胆的开拓,无疑为高职教材建设提供了良好的展示平台。

任何新生事物都有其优点与缺点,但要看事物的总体发展方向。经过不断的完善和高职教育战线上同仁们的支持,相信在不久的将来会涌现出一批符合高职教育的系列化教材,为提高高职教学质量、培养出合格的高职专业人才作出贡献。

温州职业技术学院计算机系主任
浙江省高职教育计算机类专业指导委员会副主任委员
李永平

前　言

随着 Internet 技术的飞速发展，Web 技术应用日益广泛。Microsoft 公司推出的新一代 ASP.NET 技术融合 Visual Studio 开发环境，使得 Web 开发架构更加高效简捷，ASP.NET 因此成为当前 Web 开发的主流技术和众多网络编程开发人员的首选。

本教程是与高职高专专业课程体系、培养模式改革相配套的教材。本书具有边做边学、工学结合的鲜明特色，全书以实际项目为主线，以任务实例为载体，主要围绕着 C# 编程、ASP.NET 控件和系统对象、数据库开发等几个方面展开。

本书设计的实际项目案例是“0931 网站”。它是一个以计算机软件技术专业 0931 班同学为开发设计主体做虚拟背景的门户网站。0931 网站项目包括一个三层架构的新闻中心、一个三层架构的博客网站、一个基于系统对象设计的聊天室、一个基于 XML 技术的留言板、一个提供电话区号查询的 Web 服务和一个查询发布天气预报的 Web 服务应用实例、一个基于 AJAX 技术的新闻展示改进方案以及网站后台管理、网站配置和安全性策略等。由于对这个项目进行了解构与分离，分任务分模块安排章节，因此这样的项目化教程符合项目教学、任务驱动的高职高专课程体系改革的初衷和目标。

本书内容共分 10 章。

第 1 章 ASP.NET 概述　介绍 ASP.NET 开发环境。包括.NET 框架结构与 ASP.NET 的特色优势、Visual Studio 集成开发环境、基于 C# 编程的.NET 代码后置技术和事件驱动机制，以及 ASP.NET 的基本控件与属性窗口的使用。

第 2 章使用站点导航控件和母版页　介绍 ASP.NET 站点导航系统与使用母版页整合公共元素统一布局。包括 web.sitemap 站点地图文件、SiteMapPath、TreeView 站点导航控件以及 0931 项目导航系统与母版页。

第 3 章系统对象与数据传递　介绍 Page、Request、Response、Cookie、Session 和 Application 等系统对象以及数据在页内和页间传递、记录的方法，构建了一个基于系统对象的网站聊天室。

第 4 章服务器控件和第三方控件　介绍验证控件和验证码控件、日

历控件和 JS 版日历控件以及使用 FCKeditor 在线文本编辑控件等。

第 5 章使用 ADO.NET 访问数据库　介绍三层结构设计模式、基于三层结构的系统基本框架搭建；利用数据库信息实现实体类以及项目数据访问层与业务逻辑层的实现；使用 SqlConnection、SqlCommand 和 SqlDataReader 对象及方法连接、访问数据库；ObjectDataSource 等各种数据源控件与数据绑定控件的类型和作用，以及使用 GridView、DropDownList、DetailsView、FormView 控件显示、添加、删除、编辑、查询数据等。最终创建一个可以支持分类浏览、关键字查询、后台管理等功能的多角色三层架构新闻网站。

第 6 章深入数据库编程　介绍 DataList 控件的使用以及新闻速览数据访问层与业务逻辑层的实现；使用 PagedDataSource 对象给数据绑定控件增加分页功能；使用 Repeater 控件自定义模板显示新闻等。

第 7 章 XML 访问技术　介绍 ASP.NET 访问 XML 的常用处理类以及使用 XmlDataSource 数据源控件和数据绑定控件访问 XML 的方法，构建了一个基于 XML 技术的站点留言板。

第 8 章.NET Web 服务　介绍 Web 服务的概念、创建和使用 Web 服务的方法；提供了一个返回 DataSet 对象的查询电话区号的 Web 服务实例以及在站点中使用 Web 服务查询发布天气预报的应用实例。

第 9 章网站部署与安全性配置　创建了一个可以浏览、发表用户日志和评论，支持用户及日志文章后台管理的三层架构的博客网站，并以此为载体介绍网站部署与安全性配置策略、应用程序配置文件与结构以及网站的安全认证与授权等。

第 10 章 ASP.NET AJAX　介绍 ASP.NET AJAX 的原理、ASP.NET AJAX Extensions 的安装，对 ASP.NET AJAX Extensions 的 5 大控件进行了详细的阐述，然后在此基础上对新闻网站的新闻展示提出一个基于 ASP.NET AJAX 模式的改进方案。

对于项目任务中有待深入和扩展的部分，本书以“思考与练习”的方式在习题部分提出问题并给予提示。为方便教师教学，还随本书一起提供了与本教材配套的所有实例源代码和 PPT 课件，可以登录清华大学出版社网站(www. tup. tsinghua. edu. cn)下载使用。本书所有源代码均在 Visual Studio 2005 中调试通过。

本教程定位于高职高专层次教材，在兼顾学科完整性和学术价值的同时，突出了实践性。本书可作为高职高专院校计算机相关专业学习动态网站开发的教材和各类培训学校的教材，对于在.NET 框架下的 Web 开发人员也具有一定的参考价值。

本书由程琪、张白桦编著。书中第 1～4、7～9 章由程琪编写，第 5、6、10 章由张白桦编写，全书由程琪统稿并撰写前言。许倩倩为本书网站项目设计了 Banner 图片，白雪冰、席先杰为本书的撰写提供了宝贵的意见和无私的帮助，在此一并表示诚挚的感谢！

由于时间仓促，书中疏漏、不足之处在所难免，敬请读者批评指正。

编　者

2010 年 1 月

目　录

第 1 章　ASP.NET 概述

任务 1.1　构建 ASP.NET 开发环境

任务目标

（1）了解.NET 框架结构与 ASP.NET 的特色优势。

（2）了解开发 ASP.NET 应用程序所需的基本环境和 Visual Studio 集成开发环境。

1.1.1　.NET 框架与 ASP.NET

ASP.NET 是一种动态网页技术，它提供了一个基于 Microsoft.NET 框架的 Web 开发平台。ASP.NET 2.0 支持的 Web Form 事件驱动的编程机制、它的代码后置技术以及它丰富的控件库，为构建 B/S 模式的、动态交互的 Web 应用程序系统提供了一个友好、简洁、快捷、高效的开发编程环境。ASP.NET 因此成为新一代 Web 开发的主流技术。

.NET 框架(.NET Framework)是支持 ASP.NET 应用程序的基础平台，它为.NET 应用程序提供核心服务，它由公共语言运行库(Common Language Runtime，CLR)和.NET Framework 类库组成。

公共语言运行库为多种语言的程序代码提供了编译运行环境。

.NET Framework 类库包含有上百个面向对象类。其中的常用类如 System 类、System. Data 类、System. Web 类、System. Xml 类、System. IO 类等提供了包括基本服务在内的数据访问、XML Web 服务、安全性配置、XML 访问、输入/输出等操作。

开发 ASP.NET Web 应用程序系统需要的基本环境分别有以下两种搭建方式。

（1）安装 Web 服务器 IIS、设置虚拟目录、安装 Microsoft .NET Framework 2.0 以及安装代码编辑器。

（2）使用 Visual Studio 集成开发环境。

1.1.2　安装 Web 服务器 IIS

IIS(Internet Information Server)是 Microsoft 开发的 Web 服务器。我们可以把 ASP.NET 应用程序部署到 IIS Web 服务器上发布。安装 Internet 信息服务的步骤如下：

（1）将 Windows Server 2003 操作系统光盘插入光驱。

（2）打开“控制面板”窗口，双击“添加或删除程序”选项，在弹出的“添加或删除程序”对话框中，单击“添加/删除 Windows 组件”按钮。

（3）在弹出的“Windows 组件向导”对话框中，选中 “应用程序服务器”复选框，单击“详细信息”按钮。

（4）在弹出的“应用程序服务器”对话框中，选中 ASP.NET 复选框和“Internet 信息服务(IIS)”复选框，并单击“详细信息”按钮。

（5）在弹出的“Internet 信息服务(IIS)” 对话框中，可以选择相关的组件和服务，也可以采用默认设置。

（6）单击“确定”按钮，再单击“下一步”按钮，即可以完成安装。

注意：Windows XP 与 Windows Server 2003 安装 IIS 的步骤有些不同。Windows XP 下安装的是 IIS 5.1 版，安装步骤会简单一些；而 Windows Server 2003 安装的是 IIS 6.0 版，安装时会多一些配置选项。

IIS 安装完成后，自动生成 C:\inetpub\wwwroot 目录，它是 IIS 的默认发布目录。

1.1.3 安装 Microsoft .NET Framework

Microsoft .NET Framework 2.0 包括公共语言运行库和.NET Framework 类库，读者可以进入微软官方网站(http://www.microsoft.com/express/download/ default.aspx)页面，并在该页面上单击 Download the Microsoft Web Platform 按钮，在随后打开的页面左下方单击“技术 .NET Framework”按钮进入.NET Framework 下载页面。

代码编辑器的安装、选择也非常简单：既可以是 Dreamweaver，也可以是任何文本编辑器如 FrontPage、记事本等。

1.1.4 设置虚拟目录

（1）将一个包含若干 Web 页面文件的站点文件夹比如 Chap1(ASP.NET 应用程序)部署在 IIS 的发布目录 wwwroot 下。

（2）在“控制面板”窗口双击“管理工具”选项，在打开的“管理工具”窗口中，选择“Internet 信息服务(IIS)管理器”选项。

（3）创建虚拟目录。在打开的“Internet 信息服务(IIS)管理器”窗口中，右击树形目录“网站”下的 Default Web Site(默认网站)，在弹出的快捷菜单中选择“添加虚拟目录”命令，如图 1-1 所示。在弹出的“虚拟目录创建向导”对话框中，为站点指定虚拟目录名(别名)、站点的真实路径以及设置客户端对站点的访问权限。例如在此为 Chap1 站点起别名 09Web。

（4）测试运行。右击站点中的页面文件，比如 index.htm 或者 default.aspx，在弹出的快捷菜单中选择“浏览”命令，就可以调试运行了。

注意：运行时客户端浏览器 URL 为 http://localhost/09web/index.htm(default.aspx)。

这里 localhost 为 Web 服务器(IIS)所在机器的 IP 地址所对应的域名，也可以直接写 IP 地址，09Web 是站点的虚拟目录名。



图 1-1　添加虚拟目录

1.1.5　Visual Studio 集成开发环境

事实上，微软为开发 ASP.NET 应用程序提供了一个很好的开发工具：Microsoft Visual Studio 2005。Visual Studio 2005 是一个集编辑、调试、运行为一体的集成开发环境。Visual Studio 2005 在安装时会自动检测并安装.NET Framework；并在其中内置 Web 服务器(ASP.NET Development Server)。所以除了需要部署发布应用程序之外，在 Visual Studio 2005 中开发 ASP.NET 应用时，无须配置 IIS 和设置虚拟目录。Visual Studio 2005 还内置 SQL Server 2005 数据库服务器，在后续任务中将会介绍它的使用。

微软 2006 年 11 月底发布消息称最新版开发工具 Visual Studio 2005 Express Editions 一年内在微软官方网站(http://msdn.microsoft.com/vstudio/express/support/install)提供免费下载，而且可以“永久合法使用”，并且可以使用在“商业上”。所以读者可以找到 Visual Studio 2005 并自行安装。Visual Studio 2005 安装简便，在安装向导引导下可以选择默认安装，在此不再赘述。

1.1.6　小结

使用.NET Framework 类库时，要按照它的命名空间(Namespace)约定。例如在用到 SQL 数据库相关类时，要在代码开头添加如下语句：

```
using System.Data.SqlClient;
```

1.1.7　思考与练习

安装和配置 Web 服务器 IIS、安装 Microsoft .NET Framework 2.0、建立一个虚拟目

录并运行一个简单的 HTML 文件或者 ASP.NET 文件。

任务 1.2　第一个 ASP.NET 应用程序

任务目标

（1）掌握创建 ASP.NET 应用程序的方法和 Web 页面设计中的基本控件使用。

（2）了解.NET 代码后置技术以及事件驱动机制。

1.2.1　创建 ASP.NET 程序

以下通过一个简单的登录界面说明在 Visual Studio 2005 中创建 ASP.NET 应用程序的过程。该示例根据用户输入来显示相应的欢迎信息。

（1）运行 Visual Studio 2005。在菜单栏中选择“文件”→“新建”→“项目”命令。在弹出的“新建项目”对话框中，选择左侧“其他项目类型”树形目录中的“Visual Studio 解决方案”选项；在右边“Visual Studio 已安装的模板”选项区域中选择“空白解决方案”选项，这里给解决方案命名为 0931，如图 1-2 所示，然后单击“确定”按钮。

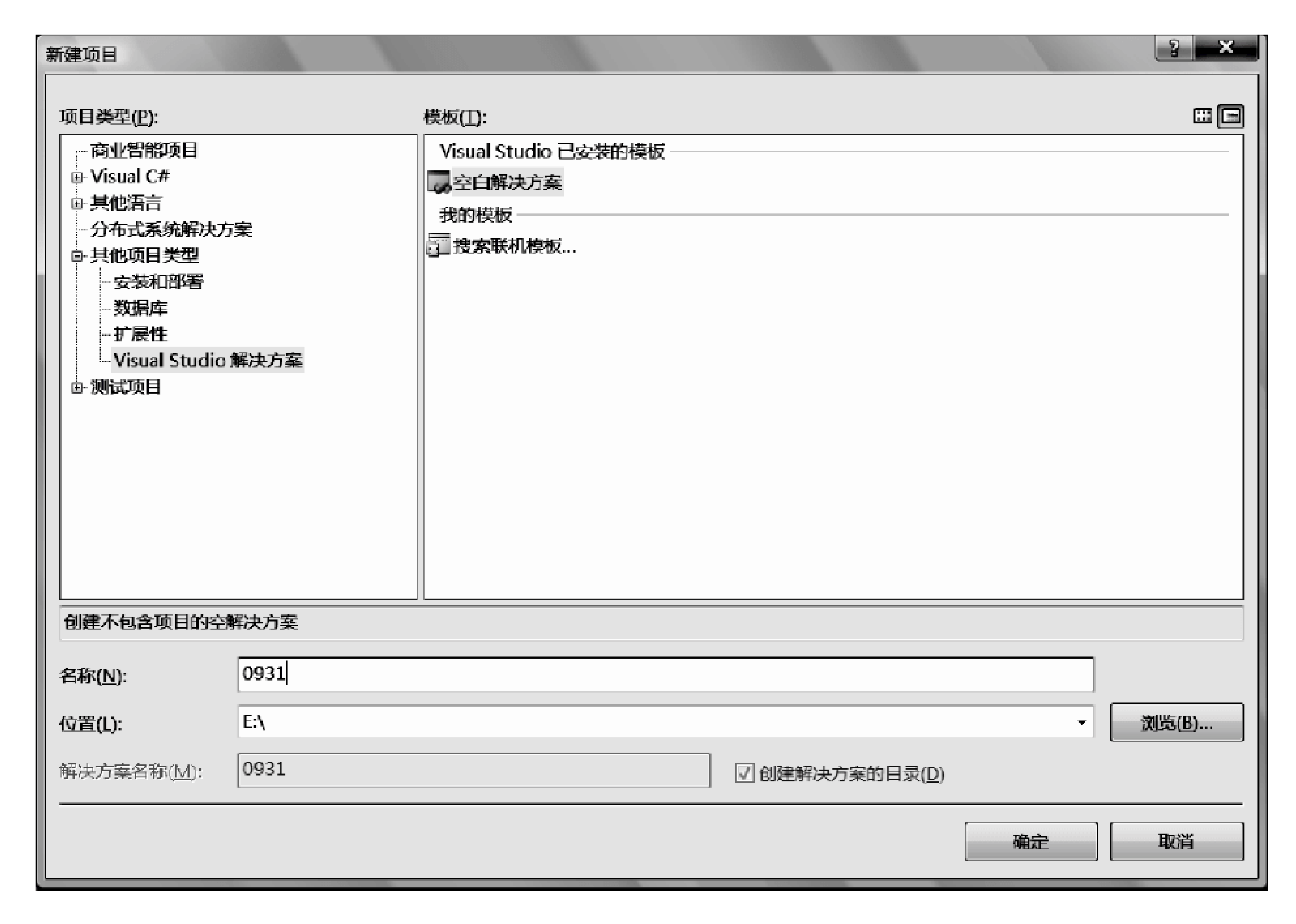


图 1-2　“新建项目”对话框

（2）在“解决方案资源管理器”窗口中，右击“解决方案‘0931’”，在弹出的快捷菜单中选择“添加”→“新建网站”命令。在如图 1-3 所示的“添加新网站”对话框中，选择“ASP.NET 网站”选项；在对话框下方的选项区域中，在“位置”下拉列表中选择“文件系统”选项，在

“语言”下拉列表中选择“Visual C#”选项；单击“浏览”按钮选择站点路径：E:\0931\Chap1，这里 Chap1 是应用程序(网站)名称。单击“确定”按钮。Visual Studio Web 窗体页面如图 1-4 所示。

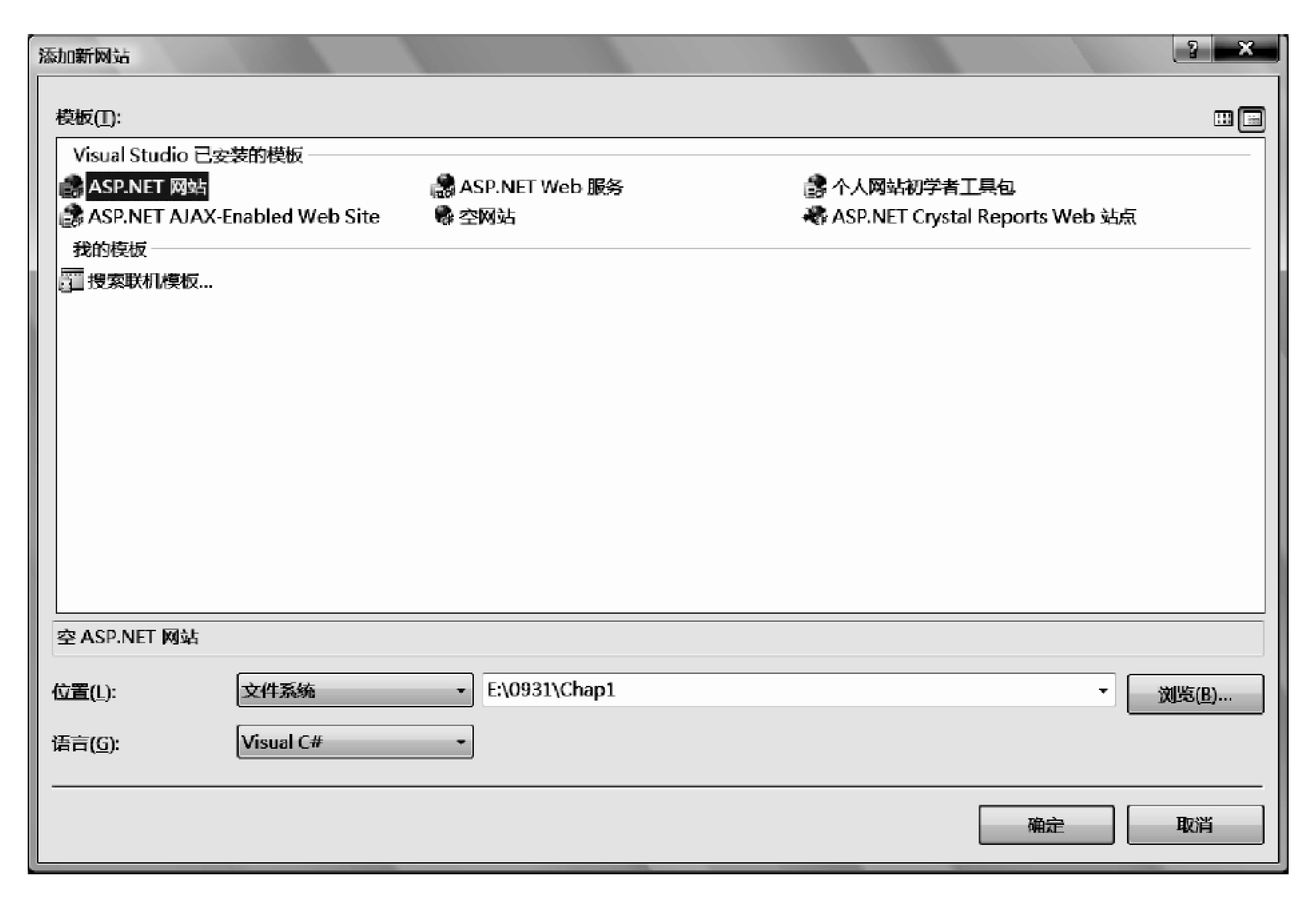


图 1-3　“添加新网站”对话框

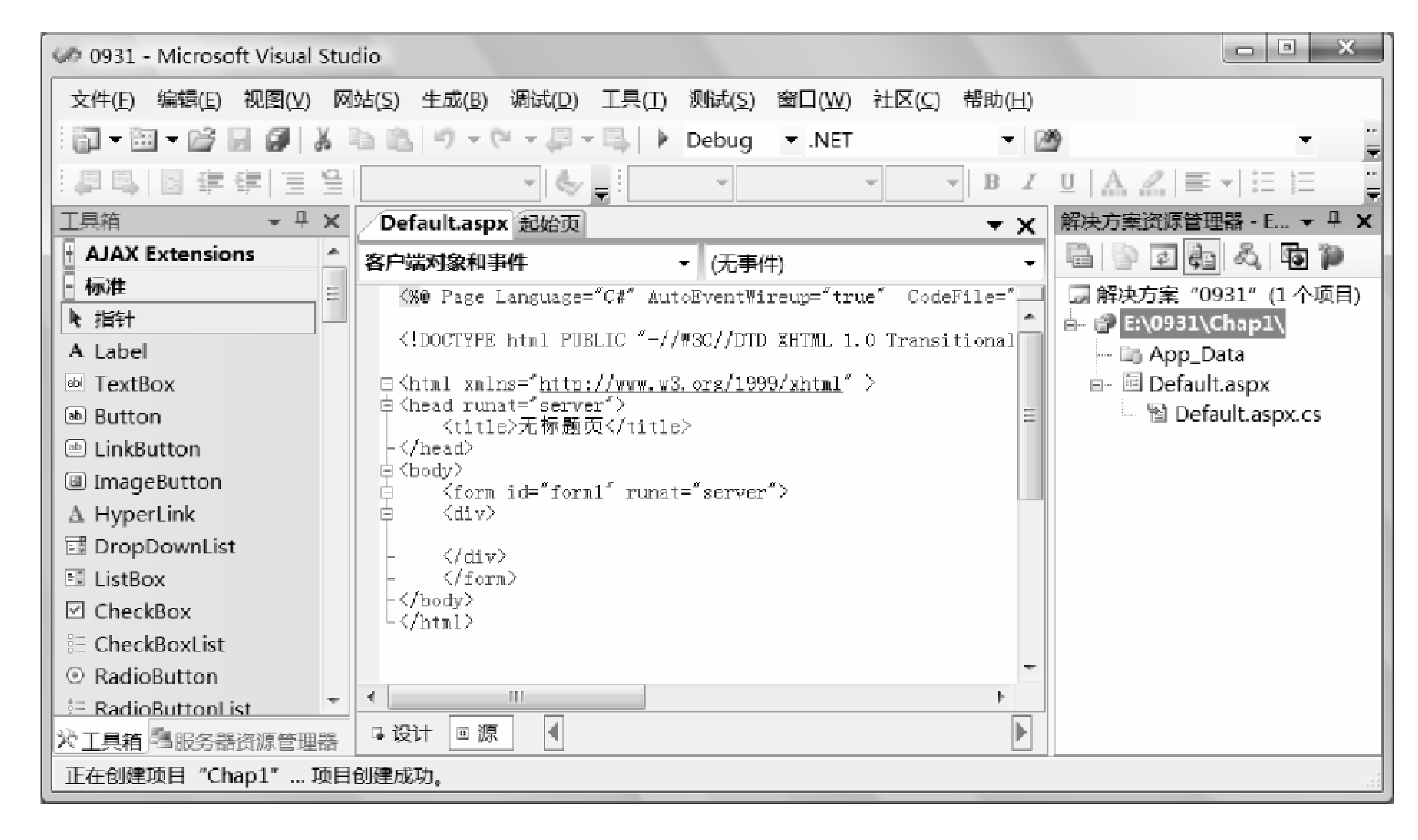


图 1-4　Visual Studio Web 窗体页面

通过 Web 窗体页面右侧的“解决方案资源管理器”面板，可以看到 Visual Studio 自动生成的内容。其中，App_Data 为应用程序的数据文件夹，Default. aspx 为一个空白的 Web 窗体页面，是网站的默认首页，Default. aspx. cs 为其代码后置文件。

将鼠标指针移动到 Web 窗体页面左侧的“工具箱”标签上，可以在展开的工具箱中看到 ASP.NET 的各类控件，添加控件时只需双击控件或者将控件拖到 Web 窗体页面上。

1.2.2 Web 窗体页面的控件设计

在 Visual Studio 2005 的“解决方案资源管理器”面板中，右击站点名 Chap1，在弹出的快捷菜单中选择“添加新项”命令。在弹出的“添加新项”对话框中选择“Web 窗体”选项，名称存为 Chap1-1. aspx，默认选择“将代码放在单独的文件中”选项，单击“添加”按钮。

切换到“设计”视图，为 Chap1-1. aspx 页面添加控件。从左侧工具箱标准组中拖出 1 个 Image 控件、3 个 Label 控件、1 个 TextBox 控件、1 个 Button 控件和 1 个 LinkButton 控件。

可以在设计视图中右击控件，在弹出的快捷菜单中选择“属性”命令，打开“属性”窗口，设置控件的属性。也可以直接在“源代码”视图中添加控件的属性和属性值。

Chap1-1. aspx 页面中添加的主要控件及属性设置如表 1-1 所示。

表 1-1 Chap1-1. aspx 页面主要控件及属性设置

控件类型	控件 ID	属性及属性值	说　明
Image	Image1	ImageUrl="～/images/qqshow. jpg"	图像控件，显示图片
Label	Label1	Text="欢迎登录 0931 网站!"	标签控件，用以显示信息
	Label2	Text="姓名"	同上
	Label4	Text=" "	留待显示与用户名对应的欢迎信息
TextBox	TextBox1	默认设置	文本框控件，输入姓名
Button	Button1	Text="确定" OnClick="Button1_Click"	单击 Button 按钮以激发 Click 事件
LinkButton	LinkButton1	PostBackUrl="～/Login/Login. aspx"	单击 LinkButton 按钮，将超链接到 Login. aspx 页面

切换到 Chap1-1. aspx 页面的“源代码”视图，可以看到如下代码：

```
<%@ Page Language = "C#" AutoEventWireup = "true" CodeFile = "Chap1-1. aspx. cs" Inherits =
"Chap1_1" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1. 0 Transitional//EN" "http://www. w3. org/TR/
xhtml1/DTD/xhtml1-transitional. dtd">

<html xmlns = "http://www. w3. org/1999/xhtml">
<head runat = "server">
    <title>第一个 ASP.NET 程序</title>
</head>
```

```
<body>
    <form id = "form1" runat = "server">
      <div>
         <asp:Image ID = "Image1" runat = "server" ImageUrl = "~/images/qqshow.jpg" Height =
"72px" Width = "79px" />
         <asp:Label ID = "Label1" runat = "server" Text = "欢迎登录 0931 网站!"></asp:Label>
         <br />
         <asp:Label ID = "Label2" runat = "server" Text = "姓名"></asp:Label>
         <asp:TextBox ID = "TextBox1" runat = "server"></asp:TextBox>
         <br />
         <asp:Button ID = "Button1" runat = "server" Text = "确定" OnClick = "Button1_Click" />
         <asp:LinkButton ID = "LinkButton1" runat = "server" PostBackUrl = "~/Login/
Login.aspx ">注册新用户</asp:LinkButton>
         <br />
         <asp:Label ID = "Label4" runat = "server" Text = ""></asp:Label>
      </div>
    </form>
</body>
</html>
```

第一行代码中,@Page 指令为 ASP.NET 页面文件指定解析和编译页面时使用的属性和值。每一个.aspx 页面文件只能包含一条@Page 指令。其中,@Page 指令的 AutoEventWireup 属性的默认值为 true,表示将自动调用页面事件;CodeFile 指定了与页面相关的后置代码文件;Inherits 属性定义了供页面继承的代码后置的类。

注意:PostBackUrl="~/Login/Login.aspx "中的"~"表示应用程序的根目录。

1.2.3 事件处理与代码后置

在 Chap1-1.aspx 页面的"设计"视图中双击 Button1 控件,可以为 Button1 控件自动添加一个属性和属性值:OnClick="Button1_Click"。

在 Chap1-1.aspx.cs 后置代码文件的 Button1_Click 事件中编写代码如下:

```
public partial class Chap1_1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
      string hello = TextBox1.Text.Trim() + "同学,欢迎你!";
      Label4.Text = hello;
    }
}
```

注意:"+"在这儿是字符串连接符。也可以用 string.Format()方法改写上面两行代码。

```
string hello = string.Format("{0}同学,欢迎你!", TextBox1.Text.Trim());
Label4.Text = hello;
```

string.Format()方法以参数的形式完成字符串之间、特别是字符串与表达式之间的连接。在本书后续章节使用 ADO.NET 访问数据库时,大多使用 Format 方法来准备 SQL 语句连接字符串,它可以简化规范代码,减少出错机会。

1.2.4 Web.config 配置文件

在标准工具栏中单击"启动调试"按钮,运行 Chap1-1.aspx。

第一次运行应用程序(网站)时,会弹出一个"未启用调试"对话框,如图 1-5 所示。选择"添加新的启用了调试的 Web.config 文件"单选按钮。单击"确定"按钮。可以在站点下看到新增的 Web.config 文件。



图 1-5 "未启用调试"对话框

Web.config 文件是放在应用程序根目录下的一个 XML 文件,它包含应用程序的配置信息,比如约定应用程序的访问规则和页面授权、存放数据库连接字符串等。本书将在第 9 章网站部署与安全性配置中详细讨论它。

Chap1-1.aspx 的运行结果如图 1-6 所示。



图 1-6 Chap1-1.aspx 的运行结果

1.2.5 常用控件与属性窗口

为了更多地说明 ASP.NET 常用控件及其属性窗口的使用,在 Chap1-1.aspx 页面中添加一个 DropDownList 控件(下拉列表框)。用户输入时可以在下拉列表框中选择来自何方。

切换到"设计"视图,为 Chap1-1.aspx 页面添加一个 Label 控件、一个 DropDownList 控件。右击 DropDownList 控件,在弹出的快捷菜单中选择"属性"命令,在打开的"属性"窗口中,选择 Items 属性,打开"ListItem 集合编辑器"对话框,单击"成员"选项区域的"添加"按钮,添加新的 ListItem 项并设置相应的 Text 属性值,如

图 1-7 所示。

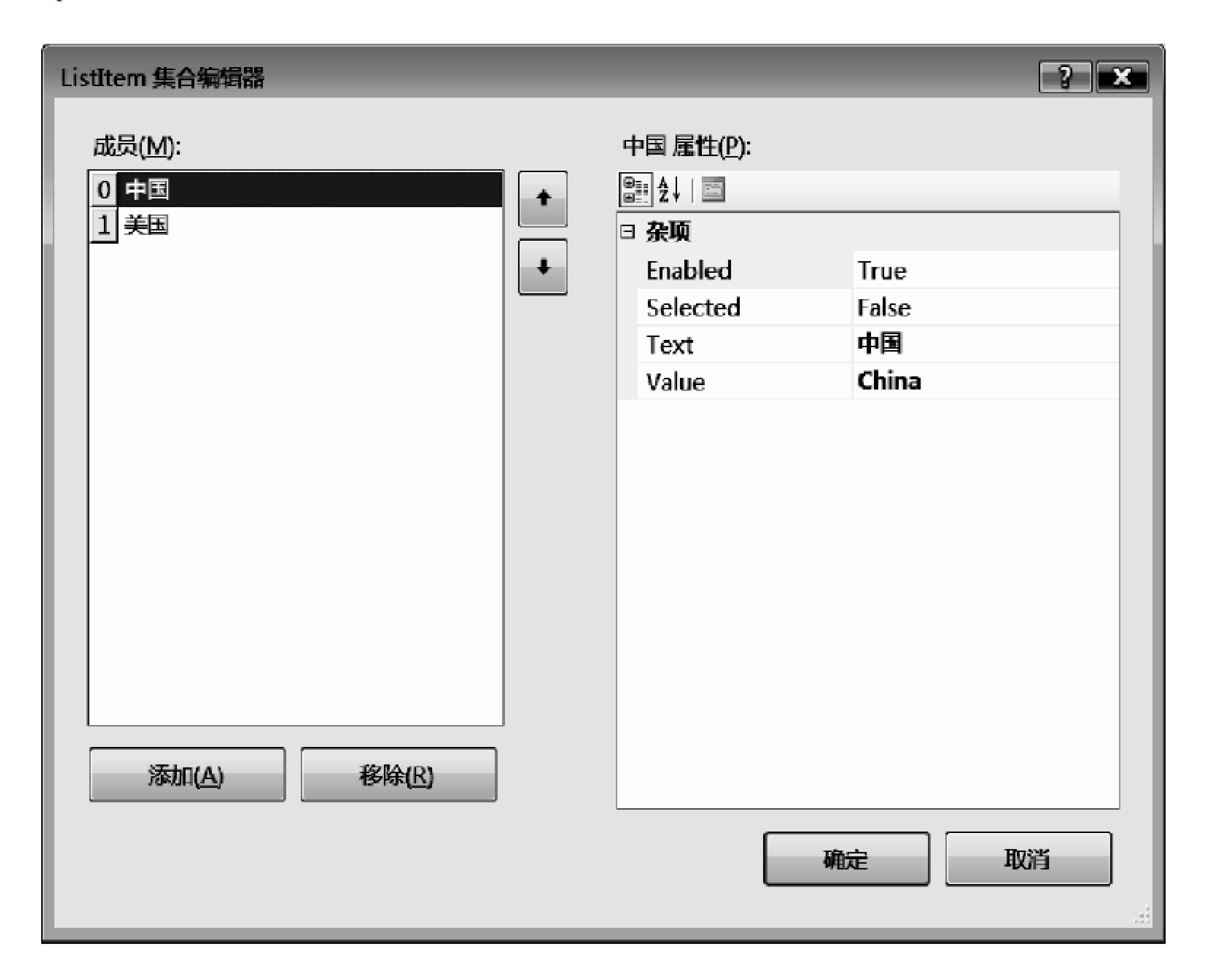


图 1-7　“ListItem 集合编辑器”对话框

在 Chap1-1.aspx 页面文件中自动生成的 DropDownList 控件代码如下：

```
<asp:Label ID="Label3" runat="server" Text="来自"></asp:Label>
<asp:DropDownList ID="DropDownList1"runat="server"OnSelectedIndexChanged=
"DropDownList1_SelectedIndexChanged">
        <asp:ListItem Value="China">中国</asp:ListItem>
        <asp:ListItem Value="America">美国</asp:ListItem>
</asp:DropDownList>
```

在“设计”视图中双击 DropDownList1 控件，打开 Chap1-1.aspx.cs 文件，然后在其 DropDownList1_SelectedIndexChanged 事件中编写代码。DropDownList 控件的 SelectedIndexChanged 事件在列表选项（索引）改变时被激活。

修改 Chap1-1.aspx.cs 文件的相关事件代码如下：

```
public partial class _Default : System.Web.UI.Page
{
    string hello = string.Empty;
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
```

```
            hello = hello + TextBox1.Text.Trim() + "同学,欢迎你!";
            Label4.Text = hello;
        }
        protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
        {
            hello = "来自" + DropDownList1.SelectedItem.Text + "的";
        }
    }
```

注意：此时 string hello =string.Empty;在类的所有方法以外定义,这是一个全局变量。

重新运行 Chap1-1.aspx。运行结果如图 1-8 所示。



图 1-8 修改后 Chap1-1.aspx 的运行结果

1.2.6 小结

(1) .NET 内置 80 多种控件,它的丰富的控件库、它的可视化设计以及基于控件+事件的简单编程方式是 ASP.NET 2.0 的特色与优势。

在.NET 2.0 中,HTML 代码和 C# 代码分别存储在 Web 窗体文件和.cs 后置文件中,称为代码后置技术。

ASP.NET 的运行机制是：当用户第一次请求.aspx 文件时,服务器引擎会编译.aspx 文件和.cs 文件,合并生成页面类并输出结果。第二次请求该页面时,就可以直接执行内存中的页面类了。

(2) 在 Visual Studio 2005 的"解决方案资源管理器"面板中,右击站点名 Chap1,在弹出的快捷菜单中选择"添加 ASP.NET 文件夹"命令,可以看到 ASP.NET 应用程序的 7 个默认文件夹。其中,Bin 文件夹通常用来存放应用程序所需的所有.dll 文件；App_Code 文件夹则可用来存放应用程序所需的.cs 等类文件。

1.2.7 思考与练习

设计一个简单的在线留言程序。页面设计及运行结果如图 1-9 和图 1-10 所示。当输入留言信息并单击"提交"按钮时,在页面下方显示留言人昵称、留言内容和提交时间。

提示：在 Web 页面后置代码文件的 Page_Load()事件中,为显示留言时间的 Label 控件设置 Text 属性值。参考代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    LabNowTime.Text = DateTime.Now.ToString();
}
```

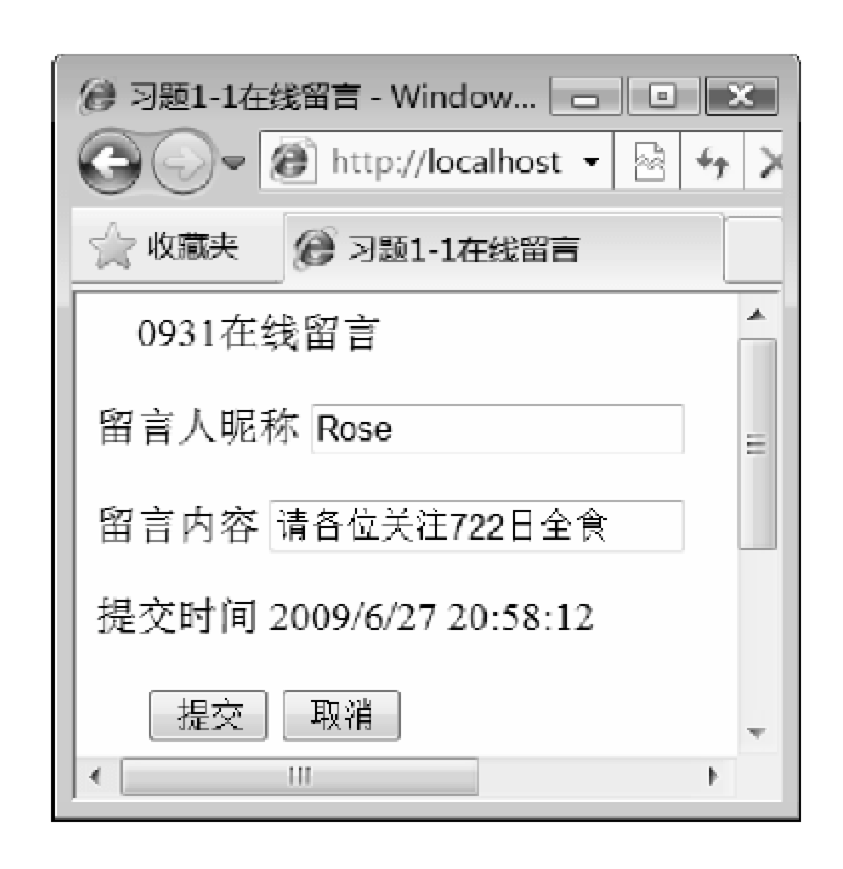


图 1-9　输入留言信息



图 1-10　提交后的结果

第 2 章　使用站点导航控件和母版页

任务 2.1　使用 SiteMapPath 设计面包屑导航

任务目标

（1）了解 ASP.NET 2.0 站点地图文件和站点导航控件在构建网站页面框架中的作用。

（2）掌握 web.sitemap 站点地图文件和 SiteMapPath 站点导航控件的使用方法。

2.1.1　导航系统与站点地图

ASP.NET 2.0 中基于站点地图的导航系统为统一页面布局和框架、设置站点导航功能提供了高效简便的方法。ASP.NET 2.0 导航系统包括站点地图、面包屑导航、树形结构导航、动态菜单等。

站点地图（web.sitemap）是一个 XML 格式的文档，它用来表示一个应用程序（即一个站点）的各个页面之间的层次结构关系。

网易首页－新闻－体育－亚运

图 2-1　面包屑导航

如图 2-1 所示，指定页面在站点的逻辑位置的导航，称为面包屑导航（Breadcrumb Navigation）。面包屑导航可以告诉用户从首页到当前页面（页面节点）之间的路径。在童话故事“汉泽尔和格雷特尔”中，当汉泽尔和格雷特尔穿过森林时，他们在沿途走过的地方都撒下了面包屑，让这些面包屑来帮助他们找到回家的路。面包屑导航让用户明了站点页面之间的层次结构关系。帮助用户在浏览网页时可以返回，“找到回家的路”！

2.1.2　网站的面包屑导航

以下为 0931 项目设计面包屑导航条。

（1）创建 ASP.NET 应用程序。在 E:\0931 下双击 0931.sln，打开 Visual Studio 2005。在“解决方案资源管理器”面板中，右击“解决方案‘0931’”，在弹出的快捷菜单中选择“添加”→“新建网站”命令，新建 E:\0931\ Navigation 站点。

（2）创建站点地图文件（web.sitemap）。在“解决方案资源管理器”面板中，右击站点名 Navigation，在弹出的快捷菜单中选择“添加新项”命令。在弹出的“添加新项”对话框中选择“站点地图”模板，单击“添加”按钮。

注意：站点地图文件名必须为 web.sitemap 且必须置于 Web 应用程序（站点）的根

目录下。

这里是按照计算机软件技术 0931 网站项目的层次结构关系设置导航路径的。web.sitemap 参考代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<siteMap xmlns = "http://schemas.microsoft.com/AspNet/SiteMap - File - 1.0">
    <siteMapNode title = "计算机软件技术专业 0931" description = "" url = "">
        <siteMapNode title = "首页" url = "Default.aspx" description = ""/>
        <siteMapNode title = "用户登录" url = "~/Login/Login.aspx" description = "" />
        <siteMapNode title = "新闻中心" url = "News.aspx" description = "" />
        <siteMapNode title = "留言板" url = "~/Messboard/Default.aspx" description = "" />
        <siteMapNode title = "聊天室" url = "~/Chatroom/Default.aspx" description = "" />
        <siteMapNode title = "发布日志" url = "~/MicroBlog/Default.aspx" description = "" />
        <siteMapNode title = "管理中心" url = "~/Admin/Admin.aspx" description = "" />
    </siteMapNode>
</siteMap>
```

在站点地图文件中：

<siteMap>为根节点，一个站点地图有且仅有一个根节点；

<siteMapNode>为页面节点，一个节点对应一个页面；

<siteMap>根节点下有且仅有一个<siteMapNode>节点；

同一个 URL 在站点地图中只能出现一次。

(3) 为 0931 网站首页 Default.aspx 创建面包屑导航。在 Visual Studio 2005 的“解决方案资源管理器”面板中，右击站点名 Navigation，在弹出的快捷菜单中选择“添加新项”命令。在弹出的“添加新项”对话框中选择“Web 窗体”选项，名称为 Default.aspx，默认将代码放在单独的文件中，单击“添加”按钮。

切换到“设计”视图，从左侧工具箱导航组中拖出 SiteMapPath 控件。可以在页面源视图中看到以下自动生成的控件代码：

```
<asp:SiteMapPath ID = "SiteMapPath1" runat = "server">
</asp:SiteMapPath>
```

运行 Default.aspx 即可以看到面包屑导航的效果了。

以下给出首页 Default.aspx 的 DIV(层)+TABLE(表格)的简单设计布局：在页面的“页眉和导航栏层”中有“面包屑导航”，并添加了 banner 图片和 navigate 图片；navigate 图片中有矩形热区，单击这些矩形热区可以超链接到 NavigateUrl 属性指定的页面；在页面底部 Foot 层使用了 footer 图片来设计页脚；在页面中间内容层，目前暂时没有实质性内容。

Default.aspx 页面主要代码如下：

```
<form id = "form1" runat = "server">
    <!-- 页眉和导航栏层 -->
    <div class = "head_layer">
        < asp: Image ID = " Image1" runat = " server" ImageUrl = " ~/images/banner. jpg" style =
"width: 1024px; height: 112px"/>
    </div>
```

```
        <div class = "navigate_layer">
            < asp: ImageMap ID = " ImageMap1" runat = " server" ImageUrl = " ~/images/navigate.
jpg" HotSpotMode = "Navigate">
            < asp:RectangleHotSpot Left = "250" Right = "300" Top = "5" Bottom = "35"NavigateUrl =
"~/Default.aspx" />
            < asp:RectangleHotSpot Left = "350" Right = "400" Top = "5" Bottom = "35"NavigateUrl =
"~/Login/Login.aspx" />
            < asp:RectangleHotSpot Left = "450" Right = "500" Top = "5" Bottom = "35"NavigateUrl =
"News.aspx" />
         …
            </asp:ImageMap>
        </div>
        <!-- 面包屑导航层 -->
        <div class = " SiteMapPath _layer">
             <asp:SiteMapPath ID = "SiteMapPath1" runat = "server">
             </asp:SiteMapPath>
        </div>
        <!-- 页面中间内容层 -->
        <div class = "content_layer">
        <p>欢迎访问 0931 网站!</p>
        </div>
        <!-- 页脚 Foot 层 -->
        <div class = "Foot_layer">
            < asp: Image ID = " Image1" runat = " server" ImageUrl = " ~/images/Footer. jpg" style =
"width: 95px; height: 53px; margin: 4" />
        </div>
    </form>
```

注意：在页面 banner 层中，Image 控件和 ImageMap 控件的用处不同。Image 控件用来添加页面 banner 图片；ImageMap 控件用来添加 navigate 导航条图片，在导航条图片中设置了若干个矩形热区，分别用来提供超链接到“首页”、“用户登录”、“新闻中心”、“后台管理中心”等页面。

(4) 同步骤(3)创建 0931 用户登录页面/Login/Login.aspx，并为其添加面包屑导航。用户登录页面的界面设计可参照图 2-3。

(5) 分别运行 Default.aspx 和 Login.aspx 页面文件。体会 SiteMapPath 站点导航控件在不同页面中的自动定位。

值得注意的是，运行当前页面时，站点地图中一定要有与当前页相应的 URL。例如，运行 Default.aspx 时，站点地图中一定要有相对应的 URL="Default.aspx"的页面节点，否则不能显示导航路径。

(6) 为面包屑导航设置格式。修改 Default.aspx。切换到“设计”视图，右击 SiteMapPath 控件，在弹出的快捷菜单中选择“自动套用格式”命令。在弹出“自动套用格式”对话框中选择方案为彩色型。

Default.aspx 页面、Login.aspx 页面的运行结果分别如图 2-2 和图 2-3 所示。

2.1.3 将 SiteMapPath 的分隔符设置为图片

SiteMapPath 控件的 PathSeparator 属性可以设置导航路径的分隔符，默认分隔符为



图 2-2　运行网站首页

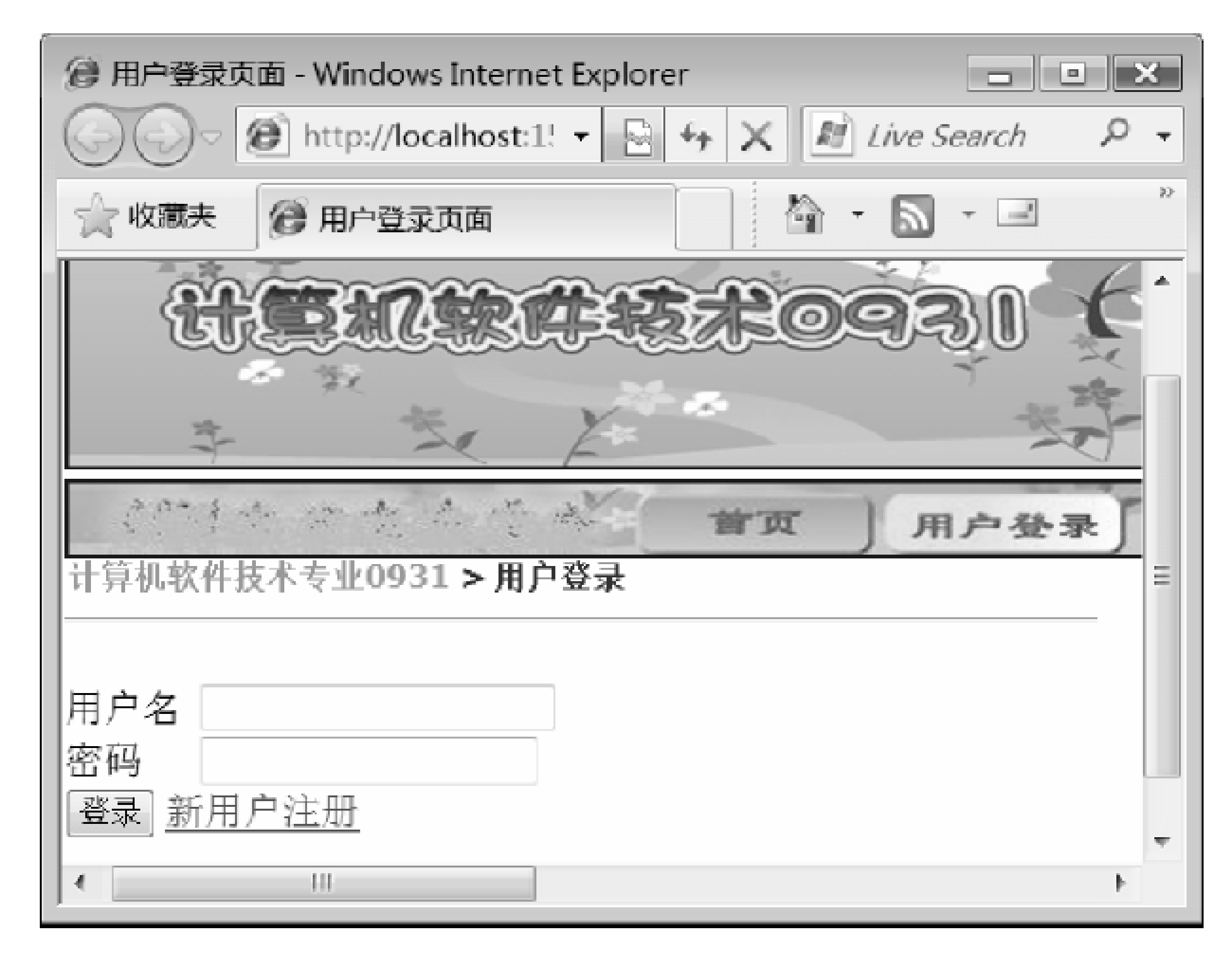


图 2-3　运行用户登录页面

“＞”。导航路径的分隔符也可以设置为图片，设置方法如下。

(1) 切换到“设计”视图，右击 SiteMapPath 控件，在弹出的快捷菜单中选择“编辑模板”→PathSeparatorTemplate 命令。

在 SiteMapPath1 PathSeparatorTemplate 模板中拖入 Image 控件，右击 Image 控件，在弹出的快捷菜单中选择“属性”命令，打开“属性”窗口。设置 ImageUrl＝"～/images/Separator1.jpg"属性值。Separator1.jpg 是分隔符图片。

Default.aspx 页面自动生成的源代码如下：

```
<form id = "form1" runat = "server">
<div>
```

```
    <asp:SiteMapPath ID = "SiteMapPath1" runat = "server" Font - Names = "Verdana" Font - Size =
"0.8em">
          <PathSeparatorStyle Font - Bold = "True" ForeColor = "#990000" />
          <CurrentNodeStyle ForeColor = "#333333" />
          <NodeStyle Font - Bold = "True" ForeColor = "#990000" />
          <RootNodeStyle Font - Bold = "True" ForeColor = "#FF8000" />
          <PathSeparatorTemplate>
              <asp:Image ID = "Image1" runat = "server" ImageUrl = "~/images/Separator1.jpg" />
          </PathSeparatorTemplate>
      </asp:SiteMapPath><br />
          欢迎访问 0931 网站!
    </div>
</form>
```

(2) 重新运行 Default.aspx,可以看到分隔符是图片时的运行结果。

2.1.4 小结

SiteMapPath 控件只能绑定站点地图,即只能使用站点地图文件作为数据源。一个站点只能有一个站点地图且必须位于应用程序(站点)的根目录下。

SiteMapPath 控件的 PathSeparatorStyle 属性可以设置导航路径分隔符的样式;SiteMapPath 控件的 ParentLevelsDisplayed 属性可以控制导航显示的级数,默认属性值是－1,表示无限制。

2.1.5 思考与练习

为 0931 网站后台管理中心设计首页和面包屑导航。

任务 2.2 使用 TreeView 设计树形结构导航

任务目标

(1) 了解 TreeView 站点导航控件的两种数据源类型及其绑定数据源的方法。
(2) 掌握 TreeView 站点导航控件的使用方法。

2.2.1 TreeView 站点导航控件

ASP.NET 2.0 提供的 TreeView 站点导航控件可以轻松设计树形结构导航系统。TreeView 控件仍可以使用站点地图(web.sitemap)作为数据源,也可以自行编写简单的 XML 文件作为 TreeView 控件的数据源。

2.2.2 网站树形目录导航

在 E:\0931 目录下双击 0931.sln 文件,打开网站 E:\0931\Navigation。

(1) 创建站点地图文件(web.sitemap)。继续使用任务 2.1 中创建的站点地图文件,

并且在新闻中心页面节点(<siteMapNode>)下增加了两个页面节点。web. sitemap 参考代码修改如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< siteMap xmlns = "http://schemas.microsoft.com/AspNet/SiteMap - File - 1.0">
   < siteMapNode title = "计算机软件技术 0931" description = "" url = "">
      < siteMapNode title = "首页" url = "Default.aspx" description = ""/>
      < siteMapNode title = "用户登录" url = "~/Login/Login.aspx" description = "" />
      < siteMapNode title = "新闻中心" url = "~/News/News.aspx" description = "">
         < siteMapNode title = "国际要闻" url = "~/News/Internat.aspx" description = "" />
         < siteMapNode title = "国内要闻" url = "~/News/Internal.aspx" description = "" />
      </siteMapNode>
      < siteMapNode title = "留言板" url = "~/MessBoard/Default.aspx" description = "" />
      < siteMapNode title = "聊天室" url = "~/ChatRoom/Default.aspx" description = "" />
      < siteMapNode title = "发布日志" url = "~/MicroBlog/Default.aspx" description = "" />
      < siteMapNode title = "管理中心" url = "~/Admin/Admin.aspx" description = "" />
   </siteMapNode>
</siteMap>
```

(2) 使用 TreeView 控件为 0931 网站各层次页面设计树形结构导航。以 News 文件夹下 Internat. aspx 国际要闻页面为例。

在“解决方案资源管理器”面板中，右击“News 文件夹”，在弹出的快捷菜单中选择“添加新项”命令，创建 Internat. aspx Web 页面文件。

切换到“设计”视图。从左侧工具箱导航组中拖出 TreeView 控件。右击 TreeView 控件，在弹出的快捷菜单中选择“显示智能标记”命令，弹出“TreeView 任务”对话框，在“选择数据源”选项区域中选择“新建数据源”选项。在弹出的“数据源配置向导”对话框中，选择“站点地图”选项，并为数据源指定 ID 为：SiteMapDataSource1(默认值)。

Internat. aspx 中与 TreeView 控件相关的关键代码如下：

```
< form id = "form1" runat = "server">
…
< div >
    < asp:TreeView ID = "TreeView1" runat = "server" DataSourceID = "SiteMapDataSource1">
    </asp:TreeView >
    < asp:SiteMapDataSource ID = "SiteMapDataSource1" runat = "server" />
</div >
…
```

这里，DataSourceID 是 TreeView 控件的属性，SiteMapDataSource1 是数据源的 ID。DataSourceID＝"SiteMapDataSource1"就是 TreeView 控件绑定数据源的方式。

(3) 为树形目录导航设置格式。在“设计”视图中，右击 TreeView 控件，在弹出的快捷菜单中选择“自动套用格式”命令。在弹出的“自动套用格式”对话框中选择方案为“新闻(型)”。

运行 Internat. aspx，运行结果如图 2-4 所示。



图 2-4　国际要闻页面树形目录导航

2.2.3　在树形目录导航中重定向页面

TreeView 控件有一个很重要的事件 SelectedNodeChanged。当用户在树形目录导航中选择或者是单击页面节点时，就会激活该事件。

例如，在 Internat.aspx 国际要闻页面单击“用户登录”节点，就可以重定向到用户登录页面。设计及编程方法如下：

(1) 打开 Internat.aspx 页面，切换到“设计”视图。

(2) 双击 TreeView 控件，在 Internat.aspx.cs 文件的 TreeView1_SelectedNodeChanged() 事件中如下编写代码：

```
protected void TreeView1_SelectedNodeChanged(object sender, EventArgs e)
{
    for (int i = 0; i < TreeView1.CheckedNodes.Count; i++)
    {
            string strURL = TreeView1.CheckedNodes[i].Text;
            Response.Redirect(strURL);
     }
}
```

(3) 运行 Internat.aspx，运行结果如图 2-4 和图 2-5 所示。

2.2.4　选择 XML 文件作为数据源

在“数据源配置向导”对话框中，可以看到 TreeView 控件可使用的数据源类型有两种：XML 文件和站点地图文件。在不需要树形导航列出全部页面目录的情况下，可以使用 XML 文件作为 TreeView 控件的数据源。

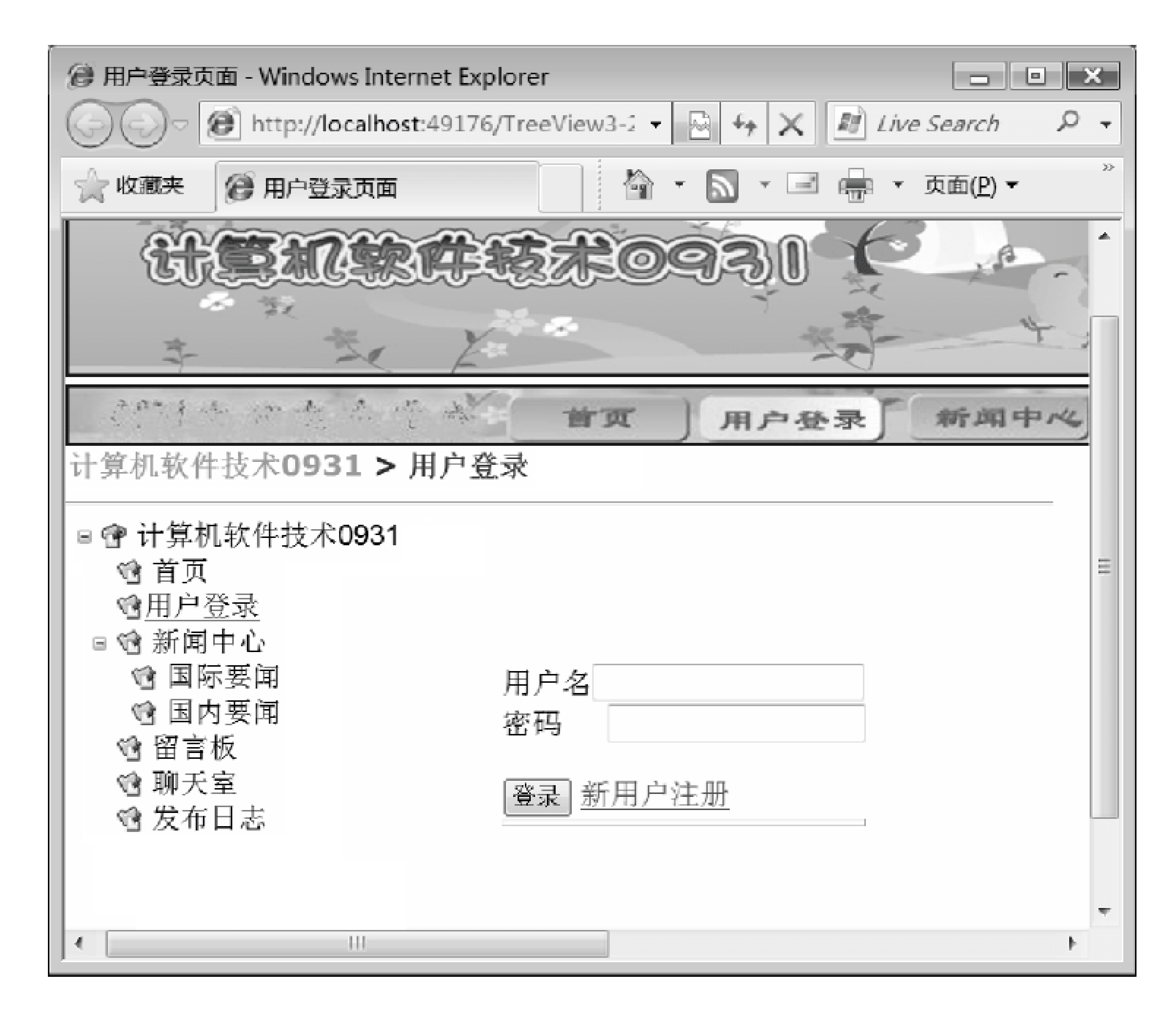


图 2-5　单击树形导航页面节点后的重定向页面

以下为 0931 后台管理中心设置树形结构导航，并说明 XML 文件作为数据源的使用方法。

(1) 创建 XML 文件(Manage. xml)。在 Visual Studio 2005 的“解决方案资源管理器”面板中，右击站点名 Navigation，在弹出的快捷菜单中选择“添加新项”命令，在弹出的“添加新项”对话框中选择 XML 文件。可以从站点 web. sitemap 中取出一部分作为 0931 后台管理中心的 XML 代码。Manage. xml 文件参考代码如下：

```
<?xml version = "1.0" encoding = "utf - 8" ?>
<siteMapNode title = "后台管理中心" url = "~/Admin/Admin.aspx" description = "" >
  <siteMapNode title = "用户管理" url = "~/Admin/UserManage.aspx" description = "">
    <siteMapNode title = "用户角色管理" url = "~/Admin/Role.aspx" description = "" />
    <siteMapNode title = "用户状态管理" url = "~/Admin/Statu.aspx" description = "" />
  </siteMapNode>
  <siteMapNode title = "新闻管理" url = "~/Admin/NewsManage.aspx" description = "" >
    <siteMapNode title = "添加新闻管理" url = "~/Admin/AddNews.aspx" description = "" />
    <siteMapNode title = "新闻分类管理" url = "~/Admin/ClassNews.aspx" description = "" />
  </siteMapNode>
</siteMapNode>
```

(2) 使用 TreeView 控件为管理中心各层次页面设计树形结构导航。以 Admin 文件夹下 Admin. aspx 后台管理中心首页为例。

首先创建 Admin. aspx 页面文件。切换到“设计”视图，从左侧工具箱导航组中拖出 TreeView 控件。右击 TreeView 控件，在弹出的快捷菜单中选择“显示智能标记”命令，

在弹出的“TreeView 任务”对话框中，在“选择数据源”选项区域中选择“新建数据源”选项。在弹出的“数据源配置向导”对话框中，选择“XML 文件”选项，并为数据源指定 ID 为：XmlDataSource1（默认值）。

在随后弹出的“配置数据源”对话框中，单击“浏览”按钮选择数据文件。在弹出的“选择 XML 文件”对话框中选定 XML 文件，这里是 Manage.xml。

右击 TreeView 控件，在弹出的快捷菜单中选择“编辑 TreeNode 数据绑定”命令。在弹出的“TreeView DataBindings 编辑器”对话框中，为 siteMapNode 节点设置 NavigateUrlField 属性值为 url、TextField 属性值为 title，如图 2-6 所示。



图 2-6 “TreeView DataBindings 编辑器”对话框

（3）当 XML 文件作为 TreeView 控件的数据源时，在 Admin.aspx 中自动生成的、与 TreeView 控件相关的主要代码如下：

```
<form id="form1" runat="server">
…
<div>
    <asp:XmlDataSource ID="XmlDataSource1" runat="server" DataFile="~/manage.xml">
    </asp:XmlDataSource>
     <asp:TreeView ID="TreeView1" runat="server" DataSourceID="XmlDataSource1"
ImageSet="Faq">
          <DataBindings>
                <asp:TreeNodeBinding DataMember="siteMapNode" TextField="title"
NavigateUrlField="url" />
          </DataBindings>
     </asp:TreeView>
```

```
</div>
</form>
```

运行 Admin.aspx，运行结果如图 2-7 所示。

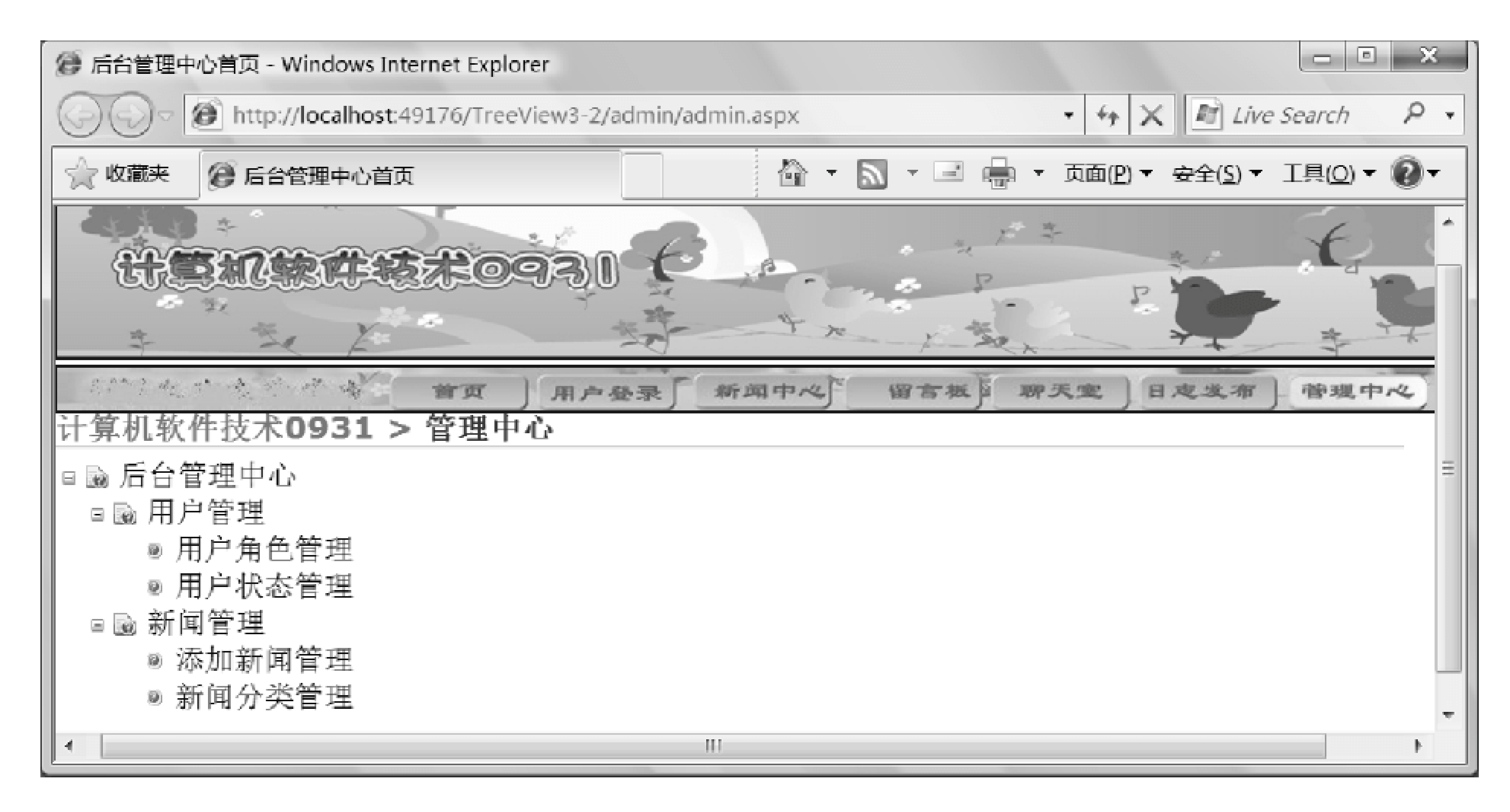


图 2-7　后台管理中心首页树形目录导航

2.2.5　小结

站点地图本身即是一个 XML 文件，在自行编写 XML 文件作为 TreeView 控件数据源时，可以参考 web.sitemap 文件的格式和内容。

2.2.6　思考与练习

在不需要树形导航列出全部页面目录的情况下，可以使用 XML 文件作为 TreeView 控件的数据源。请读者使用选择 XML 文件做数据源，为网站新闻中心设计首页和树形目录导航。

任务 2.3　母版页和导航系统

任务目标

(1) 了解母版页在整合页面公共元素、统一页面风格中的作用。

(2) 掌握创建母版页和生成内容页的方法。

2.3.1　项目概况与母版页概述

(1) 0931 网站项目包括一个多功能多角色的三层架构新闻网站、一个同样多功能的三层架构博客网站、一个基于系统对象设计的聊天室、一个基于 XML 技术的留言板、一个提供电话区号查询的 Web 服务以及网站后台管理中心等。

下面主要以新闻网站为例，说明一个网站的导航系统和母版页的设计。新闻网站前台以动态新闻显示、用户登录注册为主线，后台以用户管理、新闻管理为主要内容。

(2) 母版页是 ASP.NET 2.0 新增的一个重用技术，它可以把一些页面公共元素如网站 banner、站点导航系统甚至广告元素等整合到一个可以共享的通用页面上。

母版页为应用程序的所有页面或者一组特定页面(如新闻显示模块、后台管理模块等)提供统一的页面布局和设计风格，并且降低了应用程序开发和维护的成本。

可以把普通 Web 窗体页面设计成与母版页相对应的内容页，当请求内容页时，将整合母版页一起输出。

2.3.2 网站新闻模块母版页

下面为新闻显示模块设计母版页。

在 E:\0931 目录下双击 0931. sln 文件，运行 Visual Studio 2005，打开 E:\0931\Navigation 网站。新闻模块母版页设计步骤如下：

(1) 为面包屑导航创建站点地图。因为母版页的面包屑导航需要站点地图，所以可以继续使用任务 2.1 或任务 2.2 中创建的站点地图文件。

(2) 为新闻显示模块树形目录导航创建 XML 文件。因为在新闻模块的页面左侧只需列出与新闻相关页面的树形目录，在不需要树形导航列出全部页面目录的情况下，用 XML 文件作为 TreeView 控件的数据源。

在 Visual Studio 2005 的"解决方案资源管理器"面板中，右击站点名 Navigation，在弹出的快捷菜单中选择"添加新项"命令。在弹出的"添加新项"对话框中选择 XML 文件，命名为 News. xml 并把它放在 APP_Data 数据文件夹下。News. xml 实际上是 web. sitemap 站点地图文件中的一部分。News. xml 参考代码如下：

```
<?xml version = "1.0" encoding = "utf - 8" ?>
<newsNode title = "新闻浏览" url = " " description = " ">
   <Node title = "国际要闻" url = "~/news/Internat.aspx" description = "" >
      <subNode title = "国际要闻列表" url = "~/news/InternatList.aspx" description = "" />
      <subNode title = "国际详细要闻" url = "~/news/InternatDetail.aspx" description = "" />
   </Node>
   <Node title = "国内要闻" url = "~/news/Internal.aspx" description = "" >
      <subNode title = "国内要闻列表" url = "~/news/InternalList.aspx" description = "" />
      <subNode title = "国内详细要闻" url = "~/news/InternalDetail.aspx" description = "" />
   </Node>
   <Node title = "最新新闻" url = "/news/Latest.aspx"   description = ""/>
   <Node title = "新闻搜索" url = "/news/Search.aspx"   description = "" />
</newsNode>
```

(3) 网站的静态页面设计。准备 0931 网站静态页面公共元素如样式表文件，页眉的 banner 图片、navigate 图片，页脚的 footer 图片以及网站美工设计；并使用 HTML 的 DIV(层)+TABLE(表格)+CSS(样式表)格式进行页面布局；可将设计好的静态页面公共部分代码存成一个. html 文件，这里存为 CommonPage. html。网站静态页面公共元素可参考任务 2.1 中 Default. aspx 页面文件代码。

(4) 创建母版页。在 Visual Studio 2005 的“解决方案资源管理器”面板中，右击站点名 Navigation，在弹出的快捷菜单中选择“添加新项”命令。在弹出的“添加新项”对话框中选择母版页，扩展名为.master。这里命名为 news.master，单击“添加”按钮。news.master 中自动生成的主要代码如下：

```
<form id="form1" runat="server">
<div>
    <asp:contentplaceholder id="ContentPlaceHolder1" runat="server">
    </asp:contentplaceholder>
</div>
</form>
```

其中，ContentPlaceHolder 为内容位置控件（内容占位控件），它是母版页的专用控件，是给内容页预留位置的。以上母版页只有一个内容位置控件且命名为 ContentPlaceHolder1。一个母版页至少需要一个 ContentPlaceHolder 控件。

2.3.3 组合母版页和导航系统

把设计好的网站静态页面公共部分 CommonPage.html 以及站点导航系统（SiteMapPath 控件、TreeView 控件等）添加复制到母版页中。组合后的新闻模块母版页 news.master 参考代码如下：

```
%@ Master Language="C#" AutoEventWireup="true" CodeFile="News.master.cs" Inherits=
"News" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/
xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>一个内容占位控件的 Master</title>
    <!-- 外联样式表及内嵌样式表 -->
    <link href="App_Data/news.css" type="text/css" rel="Stylesheet" />
    <style type="text/css">
     .foot_layer{color:#606060;font-size:14px; text-align:center;}
</style>
</head>
<body>
    <form id="form1" runat="server">
        <!-- 页眉和导航栏层 -->
        <div class="head_layer">
            <asp:Image ID="Image1" runat="server" ImageUrl="~/images/banner.jpg"
Style="width: 1024px; height: 147px" />
        </div>
        <div class="navigate_layer">
             <asp:ImageMap ID="ImageMap1" runat="server" ImageUrl="~/images/
navigate.jpg" HotSpotMode="Navigate">
                <asp:RectangleHotSpot Left="250" Right="300" Top="5" Bottom="35"
NavigateUrl="Default.aspx" />
                <asp:RectangleHotSpot Left="350" Right="400" Top="5" Bottom="35"
```

```
NavigateUrl = "~/Login/Login.aspx " />
        ...
            </asp:ImageMap>
        </div>
        <!-- 面包屑导航条层 -->
        <div class = "SiteMapPath_layer">
            当前位置:<asp:SiteMapPath ID = "SiteMapPath1" runat = "server">
                    </asp:SiteMapPath>
        </div>
        <!-- 页面中间内容层,嵌套一行两列表格 -->
        <div class = "content_layer">
            <table>
                <tr>
                    <!-- 左边列,树形目录导航层 -->
                    <td>
                        <div class = "TreeView_layer">
                            <asp:TreeView ID = "TreeView1" runat = "server" DataSourceID =
"XmlDataSource1">
                                <DataBindings>
                                    <asp:TreeNodeBinding DataMember = "newsNode"Navigate-
UrlField = "url" TextField = "title" />
                                    <asp:TreeNodeBinding DataMember = "Node"NavigateUrl-
Field = "url" TextField = "title" />
                                    <asp:TreeNodeBinding DataMember = "subNode"Navigate-
UrlField = "url" TextField = "title" />
                                </DataBindings>
                            </asp:TreeView>
                            <asp:SiteMapDataSource ID = "SiteMapDataSource1"runat = "ser-
ver"/>
                            <asp:XmlDataSource ID = "XmlDataSource1"runat = "server" Dat-
aFile = "~/App_Data/news.xml">
                            </asp:XmlDataSource>
                        </div>
                    </td>
                    <!-- 右边列,内容占位控件层(预留给内容页) -->
                    <td>
                        <div class = "ContentPlaceHolder_layer">
                            <asp:ContentPlaceHolder ID = "ContentPlaceHolder1" runat =
"server">
                            </asp:ContentPlaceHolder>
                        </div>
                    </td>
                </tr>
            </table>
        </div>
        <!-- 页脚 Footer 层 -->
        <div class = "Foot_layer">
            <asp:Image ID = "Image2" runat = "server" ImageUrl = " ~/images/Footer.jpg"
Style = "width: 95px; height: 53px; margin: 4" />
```

```
        </div>
    </form>
</body>
</html>
```

2.3.4　创建内容页

可以将现有的普通 Web 窗体页面文件改造成内容页。

(1) 创建普通的 Web 窗体页面。仍以 News 文件夹下 Internat.aspx 国际要闻页面为例。右击 News 文件夹,在弹出的快捷菜单中选择"添加新项"命令,创建 Internat.aspx Web 页面文件。利用数据库生成新闻显示的动态页面要在本书的后续章节中学习,所以当前可以只创建一个简单的显示国际新闻的静态页面。

(2) 将 Internat.aspx 页面改造成内容页。

① 在@Page 标记中设置属性 MasterPageFile="news.master",并去除页面的 HTML 标记和 Form 标记。

② 创建<asp:Content>内容控件,注意内容控件必须是内容页中的顶级控件。

③ 设置内容控件的 ContentPlaceHolderID 属性。

ContentPlaceHolderID="ContentPlaceHolder1"的意思是:此内容页出现在与母版页相对应的、内容占位控件 ID="ContentPlaceHolder1"的位置中;注意这里 ContentPlaceHolder1 须与母版页中内容占位控件的命名一致。

Internat.aspx 内容页实现代码如下:

```
<%@ Page Language="C#" AutoEventWireup="true" MasterPageFile="~/news.master"
        CodeFile="internat.aspx.cs" Inherits="news_internat" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="Server">
    <div>
      <table>
        <tr><td rowspan="2"><asp:Image ID="Image1" runat="server" ImageUrl="~/
images/flood.jpg" Width="95" Height="121" /></td>
        <td style="font-size: small; width: 773px;"><a href="">澳大利亚遭暴风雨袭击
</a></td></tr>
        <tr><td style="font-size: small; width: 773px;" align="left"> 新华网 3 月
23 日报道 澳大利亚东海岸几天来连降暴雨,造成洪水泛滥,数千人被迫离开家园.昆士兰首府布
里斯班两天的降雨量已经能够为这座城市提供超过 1 年的饮用水……</td></tr>
      </table>
    </div>
</asp:Content>
```

(3) 运行内容页 Internat.aspx,运行结果如图 2-8 所示。

2.3.5　有多个 ContentPlaceHolder 控件时的母版页布局

当一个母版页需要有多个 ContentPlaceHolder 控件时,仍需要使用 HTML 的 DIV+TABLE+CSS 进行页面设计布局。下面以 0931 新闻模块母版页为例进行介绍。

(1) 参看 2.3.3 小节中的 news.master 页面代码。将 news.master 页面中间层,改



图 2-8　请求内容页时与母版页一起输出的结果

为嵌套一行三列表格，在表格的右侧第三列增加两个内容控件的位置。修改后 news.master 的内容位置控件部分代码如下：

```
<!-- 页面中间层,嵌套一行三列表格 -->
<div class = "content_layer">
    <table>
        <tr>
            <!-- 表格左边列放置树形目录导航 -->
            <td> … </td>
            <!-- 表格中间列,放置母版页的 contentplaceholder1 内容位置控件 -->
            <td style = "width: 1795px">
                <div class = " ContentPlaceHolder1_layer">
                    <asp:contentplaceholder id = "ContentPlaceHolder1" runat = "server">
                    </asp:contentplaceholder>
                </div>
            </td>
            <!-- 表格第三列分两层,分别放置母版页的 Contentplaceholder2、3 内容位置控件 -->
            <td style = "width: 476px">
                <div class = " ContentPlaceHolder3_layer">
                    <asp:contentplaceholder id = "ContentPlaceHolder3" runat = "server">
                    </asp:contentplaceholder>
                </div>
                <div class = " ContentPlaceHolder2_layer">
                    <asp:contentplaceholder id = "ContentPlaceHolder2" runat = "server">
                    </asp:contentplaceholder>
                </div>
            </td>
```

```
        </tr>
    </table>
</div>
```

（2）相应地修改新闻显示页面。仍以 Internat.aspx 页面为例，在 Internat.aspx 中增加两个<asp:Content>内容控件，增加的代码如下：

```
<asp:Content ID="Content3" ContentPlaceHolderID="ContentPlaceHolder3" Runat="Server">
    <!-- 新闻图片显示页面代码 -->
    <div>
        <asp:Image ID="Image1" runat="server" ImageUrl="~/images/ NewsPicture.jpg "
style="width: 281px; height: 126px;"/>
    </div>
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="ContentPlaceHolder2" runat="Server">
    <!-- 用户登录页面代码 -->
    <div class="Login_layer">
        <table border="1">
            <tr><td><asp:Label ID="Lab1" runat="server" Text="用户名"></asp:Label>
                    <asp:TextBox ID="txt_name" runat="server"></asp:TextBox></td>
</tr>
            <tr><td><asp:Label ID="Lab2" runat="server" Text="密码" Width="48px"
Height="19px"></asp:Label>
                    <asp:TextBox ID="txt_pw" runat="server" TextMode="Password">
</asp:TextBox></td></tr>
             <tr><td><asp:Button ID="Button1" runat="server" Text="登录" />
                     <asp:LinkButton ID="LinkButton1" runat="server" Text="新用户注
册" Height="21px" Width="100px" PostBackUrl="~/Login/NewLogin.aspx"></asp:LinkButton>
</td></tr>
        </table>
    </div>
</asp:Content>
```

（3）重新运行内容页 Internat.aspx，运行结果如图 2-9 所示。

2.3.6　小结

可以将普通 Web 窗体页面文件改造成内容页，也可以在母版页中直接添加内容页。在母版页中添加内容页有以下两种方式。

（1）在母版页中直接添加内容页。切换到“设计”视图，在 news.master 母版页任意位置右击，在弹出的快捷菜单中选择“添加内容页”命令，系统自动生成 Default.aspx 内容页框架代码如下：

```
<%@ Page Language="C#" MasterPageFile="~/news.master" AutoEventWireup="true"
         CodeFile="Default.aspx.cs" Inherits="_Default" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp:Content>
```

（2）在 Visual Studio 2005 的“解决方案资源管理器”面板中，右击站点名，在弹出的



图 2-9　有多个内容占位控件时的母版页布局

快捷菜单中选择“添加新项”命令，在添加“Web 窗体”生成 aspx 页面时选中“选择母版页”复选框，在后续弹出的选择母版页的对话框中选择需要的母版页。

2.3.7　思考与练习

为 0931 项目的留言板模块创建母版页和导航。

提示：准备留言板模块所需页面公共元素，如样式表文件，页眉的 banner 图片、navigate 图片，页脚的 footer 图片，并使用 HTML 的 DIV＋TABLE＋CSS 设计页面布局。

第 3 章　系统对象与数据传递

任务 3.1　获取用户输入信息和客户端环境信息

任务目标

（1）了解 Page、Request、Response 等系统对象的属性、方法与事件。

（2）掌握页内数据传递和页间数据传递的方法和技术。

3.1.1　ASP.NET 系统对象概述

ASP.NET 提供了一些封装类，这些封装类的实例称为系统对象。系统对象可以直接在页面上使用。ASP.NET 常用的系统对象有：用来处理页面和页面间操作的 Page 对象；用来处理页面输入/输出请求和响应的 Request 对象和 Response 对象；用来在页面之间记录、保存和传递数据的 Cookie 对象、Session 对象和 Application 对象；用来提供服务器端访问的 Server 对象等。

例如，Page 对象是 System. Web. UI. Page 类的实例，所有的 ASP.NET 页面都继承自此页面类。Page 对象包含了所有用来处理页面和页面间操作的属性、方法与事件。

3.1.2　页内数据传递

下面通过一个用户注册实例来说明 Page 对象属性的使用以及 ASP.NET 页面中数据提交与传递的方式。在本实例中，当页面首次加载时，显示注册界面；当用户提交注册信息时，将获取用户输入信息与客户端环境信息并在页面下方显示。用户注册页面界面设计以及运行效果如图 3-1 所示。其设计步骤如下：

（1）在 E:\0931 目录下双击 0931. sln 文件，打开 Visual Studio 2005。在“解决方案资源管理器”面板中，右击“解决方案‘0931’”，在弹出的快捷菜单中选择“添加”→“新建网站”命令。新建网站命名为 IsCrossPagePostBack。

右击站点名，在弹出的快捷菜单中选择“添加新项”命令。在弹出的“添加新项”对话框中选择“Web 窗体”选项，创建用户注册页面 Login. aspx。

参照图 3-1，在“设计”视图中为 Login. aspx 页面添加 3 个 TextBox 控件、1 个 DropDownList 控件、2 个 Button 控件以及若干个用来显示信息的 Label 控件。可以将上述控件用 HTML 的 DIV(层)＋TABEL(表格)加以布局。

Login. aspx 页面主要控件及属性设置如表 3-1 所示。

表 3-1 Login. aspx 页面主要控件及属性设置

控件类型	控件 ID	属性及属性值	说明
TextBox	txt_uname	默认设置	用户名
	txt_pw	TextMode="Password"	密码
	txt_answer	默认设置	安全回答
DropDownList	DropDownList_ques	默认设置	提示问题
Button	btn_OK	Text="注册" OnClick=" btn_OK _Click"	注册按钮
	btn_cancel	Text="取消"	取消按钮
Label	Label1、Label2、Label3…	默认设置	用来显示信息

Login. aspx 中与 DropDownList 控件相关的代码如下：

```
<form id="form1" runat="server">
  <div>
     <table style="width: 247px;">
      …
      <tr>
        <td>提问</td>
        <td><asp:DropDownList ID="DropDownList_ques" runat="server" Width="130px">
                <asp:ListItem>你最喜欢的食物?</asp:ListItem>
                <asp:ListItem>你最喜欢唱的歌?</asp:ListItem>
          </asp:DropDownList></td>
      </tr>
…
```

(2) 在设计视图中双击 btn_OK 按钮，在 Login. aspx. cs 文件的 btn_OK _Click 事件中编写如下代码：

```
protected void btn_OK_Click1(object sender, EventArgs e)
{
     if (Page.IsPostBack) //页内数据回传
     {
           if (this.txt_uname.Text != string.Empty && this.txt_pw.Text != string.Empty)
           {
                this.Label3.Text = "<br>用户名: " + this.txt_uname.Text
                     + "<br>密  码: " + this.txt_pw.Text
                     + "<br>安全提问: " + this.DropDownList_ques.SelectedItem.Text
                     + "<br>安全答案: " + this.txt_answer.Text;
                this. Label1.Text = "<br>来自" + Request.ServerVariables["remote_
addr"].ToString() + "的朋友,您好!";
                this.Label2.Text = "<br>您当前运行的文件是: " + Request.ServerVariables
["script_name"].ToString();
                this. Label3.Text = "<p>以下是您提交的信息,请确认!<br>" + this.Label3.
Text;
            }
        }
}
```

说明：

① Page 对象的 Page. IsPostBack 属性可以用来判断页内是否有表单数据提交。

if (Page. IsPostBack)为 true 时，表示是页内数据传递，通常称为页面回传。这里可以确保当有数据在页内提交传递时，获取并显示相关信息。

if (!Page. IsPostBack)为 true 时，表示是第一次加载页面。这里可以确保首次加载页面时，显示注册界面。

② Request 对象的 ServerVariables 属性可以用来获取服务器端和客户端的环境变量信息。例如：

```
Request.ServerVariables["remote_addr"];          //用来获取客户端的 IP 地址
Request.ServerVariables["script_name"];          //用来获取当前执行文件的路径
```

类似的用法还有：

```
Response.Write(Request.ServerVariables["http_user_agent"].ToString());
Response.Write(Request.ServerVariables["http_accept_language"].ToString());
```

可以分别用来获取并显示客户端浏览器的版本与语言。

(3) 运行 Login. aspx。如图 3-1 所示，可以看到表单数据在页内提交传递的结果。

注意：在第一次加载页面时，是显示注册界面的；而当用户提交注册信息时，才有与用户输入相关的欢迎信息与确认信息在页面下方显示。

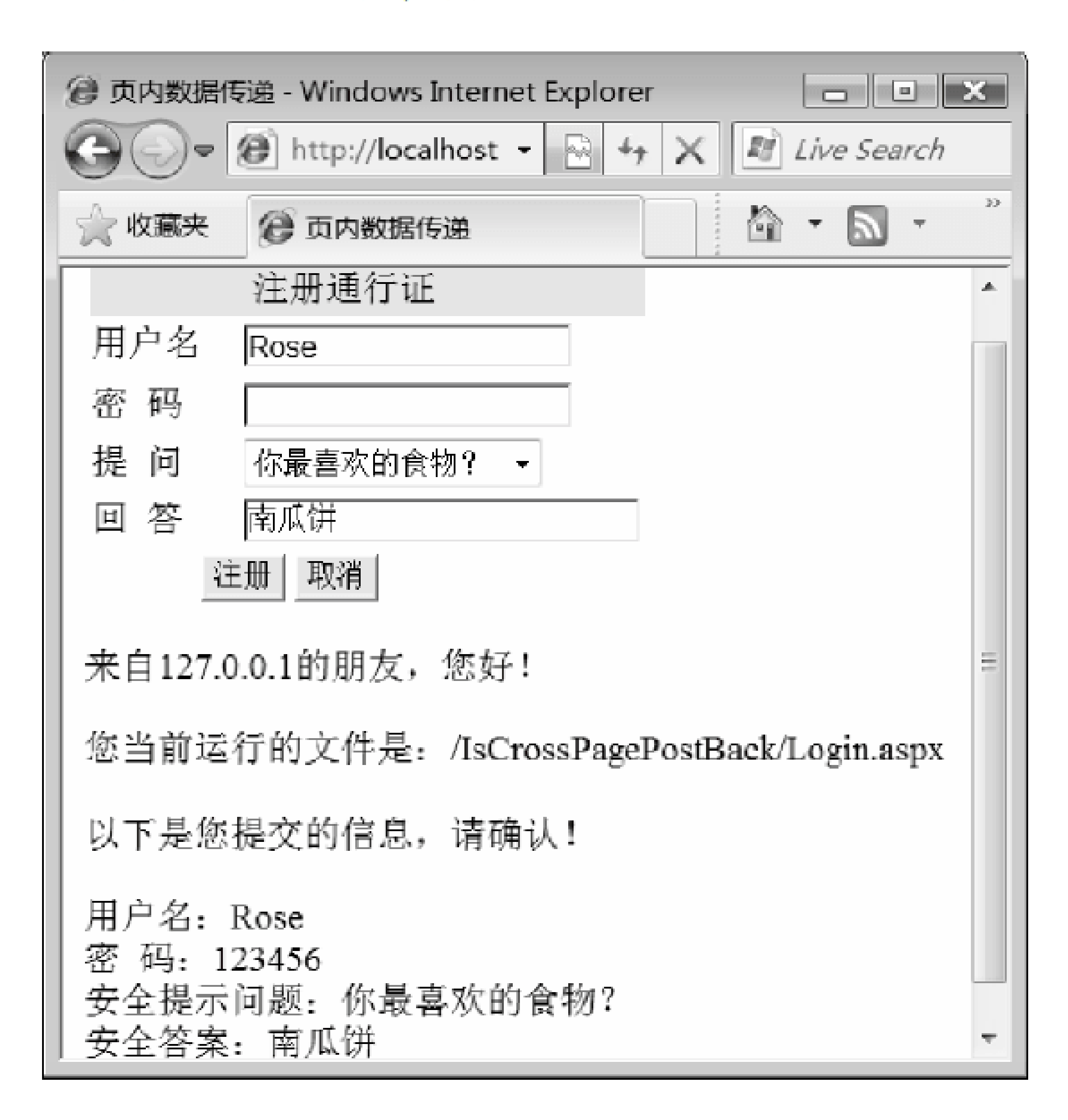


图 3-1　页内数据传递

3.1.3　跨页数据传递

下面仍然以此 Login. aspx 用户注册页面为例，说明如何在另一页上获取前一页表单提交的数据。

（1）运行 Visual Studio 2005，打开 IsCrossPagePostBack 网站。修改 Login. aspx 页面文件，设置 btn_OK 按钮的 PostBackUrl 属性。

Login. aspx 中与 btn_OK 控件相关的代码设置如下：

```
<form id="form1" runat="server">
  <div>
    <table style="width: 247px;">
    …
      <tr>
        <td colspan="2">
            <asp:Button ID="btn_OK" runat="server" Text="注册" BackColor="#COFFC0"
PostBackUrl="~/ConfirmPage.aspx" />
            <asp:Button ID="btn_cancel" runat="server" Text="取消" BackColor="#COFFC0"/>
          </td>
    </tr>
    …
    </table>
  </div>
</form>
```

PostBackUrl 属性指明本页表单数据提交传递时的获取页为 ConfirmPage. aspx。

（2）在站点下新建一个 Web 页面文件 ConfirmPage. aspx，获取并显示 Login. aspx 页面提交的信息。

在 ConfirmPage. aspx 上添加 Label1、Label2、Label3 控件用来输出用户注册信息和环境信息；添加 Button1、Button2 按钮用来确认和返回。ConfirmPage. aspx 页面设计可以参考图 3-3。

（3）在 ConfirmPage. aspx. cs 的 Page_Load 事件中编写代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //判断前面传输页是否为 null
    if (Page.PreviousPage != null)
    {
      //判断前页是否使用跨页数据提交
      if (PreviousPage.IsCrossPagePostBack == true)
      {
           if (((TextBox)Page.PreviousPage.FindControl("txt_uname")).Text != string.
Empty && ((TextBox)Page.PreviousPage.FindControl("txt_pw")).Text != string.Empty)
          {
               this.Label3.Text = this.Label3.Text + "<br>用户名：" + ((TextBox)Page.
PreviousPage.FindControl("txt_uname")).Text
              + "</br>密   码：" + ((TextBox)Page.PreviousPage.FindControl("txt_pw")).Text
              + "</br>安全提示问题：" + ((DropDownList)Page.PreviousPage.FindControl("
DropDownList_ques")).Text
              + "</br>安全答案：" + ((TextBox)Page.PreviousPage.FindControl("txt_answer"))
.Text;
                this.Label1.Text = "</br>来自" + Request.ServerVariables["remote_addr"]
.ToString() + "的朋友,您好!";
```

```
                this.Label2.Text = "</br>您当前运行的文件是: " + Request.ServerVariables
["script_name"].ToString();
                this.Label3.Text = "<p>以下是您提交的信息,请确认!</br>" + this.Label3.
Text;
            }
            else
            {
                Response.Write("<script>alert(\"必须输入用户名和密码!\")</script>");
            }
        }
    }
}
```

说明:

① 必须先用 if(Page.PreviousPage != null)确保前面的传输页对象非 null,否则程序调试时会出现"用户代码未处理 NullReferenceException"异常,提示在执行下一句代码前,检查确定对象是否为空。Page.PreviousPage 属性用来表示向当前页传输数据的页。

② if (PreviousPage.IsCrossPagePostBack == true)用来判断前一页面是否使用了跨页数据提交。

③ Page.PreviousPage.FindControl ("控件 ID")方法,用来找到并获取前面传输页控件的值。

(4) 在 ConfirmPage.aspx 页面的设计视图中分别双击 Button1(确认)按钮和 Button2(返回)按钮。

在 ConfirmPage.aspx.cs 的 Button1_Click、Button2_Click 事件中编写如下代码:

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write("<script>alert(\"已成功创建您的账户!\")</script>");
}
protected void Button2_Click(object sender, EventArgs e)
{
    Response.Redirect("Login.aspx");
}
```

说明:

① 使用 Response 对象的 Write 方法,可以将 HTTP 响应数据发送到客户端,实现页面的输出控制。这里用到了 alert()函数,它是 JavaScript 语言的输出框函数。ASP.NET 以多种形式支持 JavaScript 脚本代码。

② 使用 Response 对象的 Redirec 方法可以重定向返回注册页面。

(5) 重新运行 Login.aspx 页面文件,输入注册信息后提交并确认。运行结果如图 3-2～图 3-4 所示。



图 3-2　输入注册信息

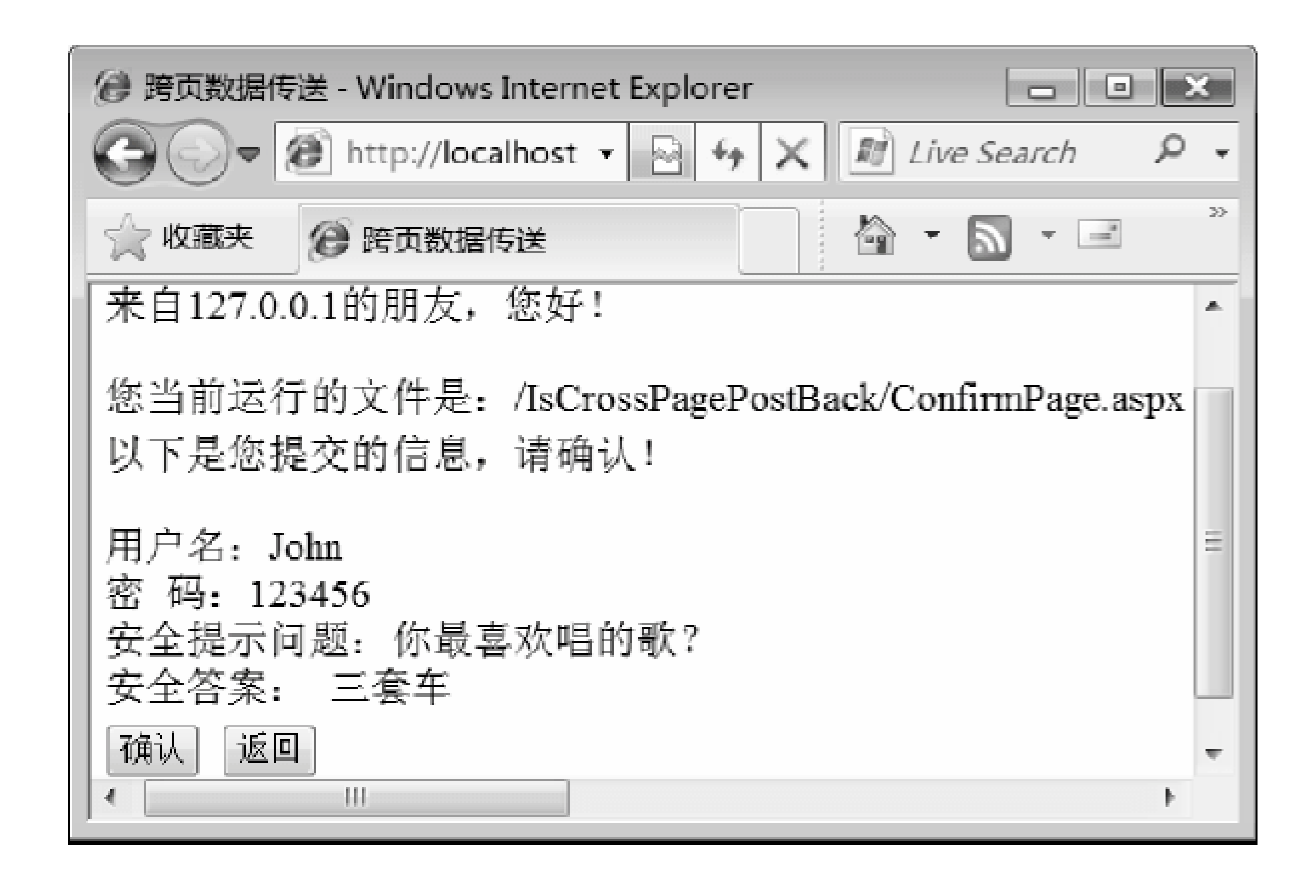


图 3-3　数据跨页传递

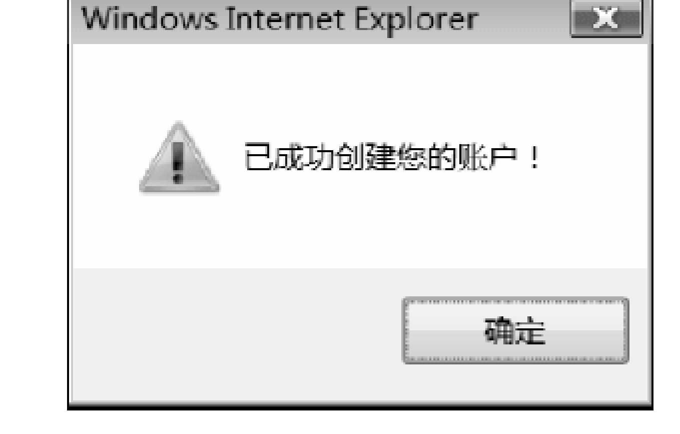


图 3-4　单击“确认”按钮的结果

3.1.4　小结

(1) Page 对象的@ Page 指令为页面指定了解析和编译页面时使用的属性和值。当页面@ Page 指令中的 AutoEventWireup="true"时，在页面装载时首先执行 Page_Load 事件过程。

(2) Request 对象常用来获取页面的输入以及用来获取客户端的环境变量信息。

(3) Response 对象用来控制页面输出，例如：

```
Response.Write(string express);     //在页面上输出信息
Response.Redirect(string url);      //重定向到另一页面
```

3.1.5　思考与练习

1. 设计一用户登录页面，要求在用户登录后在本页显示与用户名相关的欢迎信息。注意使用 Page.IsPostBack 属性，确保页内数据传递时，显示登录欢迎信息；而当页面首次加载时，显示登录界面。

2. 修改第 1 题程序，在另一页显示与用户名相对应的欢迎信息。注意使用 Page.PreviousPage 属性和 Page.IsCrossPagePostBack 属性。

任务 3.2　记录用户访问网站的时间和次数

任务目标

(1) 了解 Cookie 对象的常用属性和方法。

(2) 掌握使用 Cookie 对象记录用户登录和访问信息的方法和技术。

3.2.1　Cookie 对象简介

Cookie 对象是 System.Web.HttpCookie 类的实例。Cookie 实际上是一小段文本信息，可以由 Web 服务器写到客户端硬盘，或者从客户端读回来。可以用此方法来跟踪客

户端用户的登录和访问信息。例如：

（1）用 Response 对象的 Cookies 方法写 Cookie。

```
//创建一 Cookie 实例
HttpCookie UNCookie = new HttpCookie("UserName", "小松鼠");
//把 Cookie 信息写到客户端
Response.Cookies.Add(UNCookie);
```

（2）用 Request 对象的 Cookies 方法读 Cookie。

```
//读取客户端 Cookie 信息
string UName = Request.Cookies["UserName"].Value;
```

3.2.2　记录用户的访问信息

下面通过一个实例来说明 Cookie 对象的应用。

在本实例中，当用户第一次来访问，显示问候和首次光临本站的信息，并提示用户登录。运行结果如图 3-5 和图 3-6 所示。

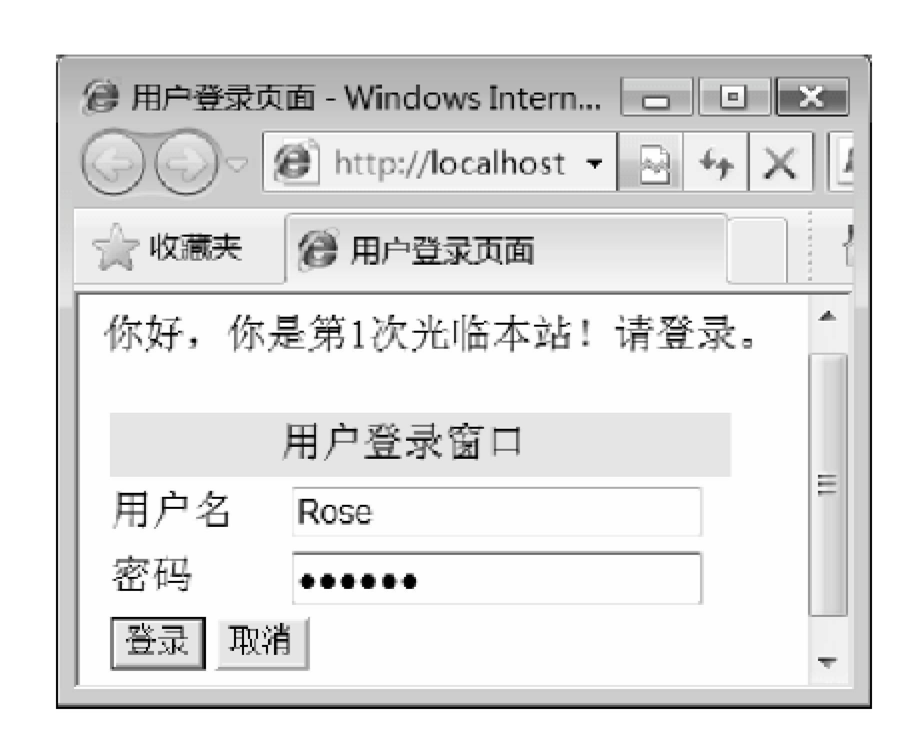




图 3-5　第一次访问时

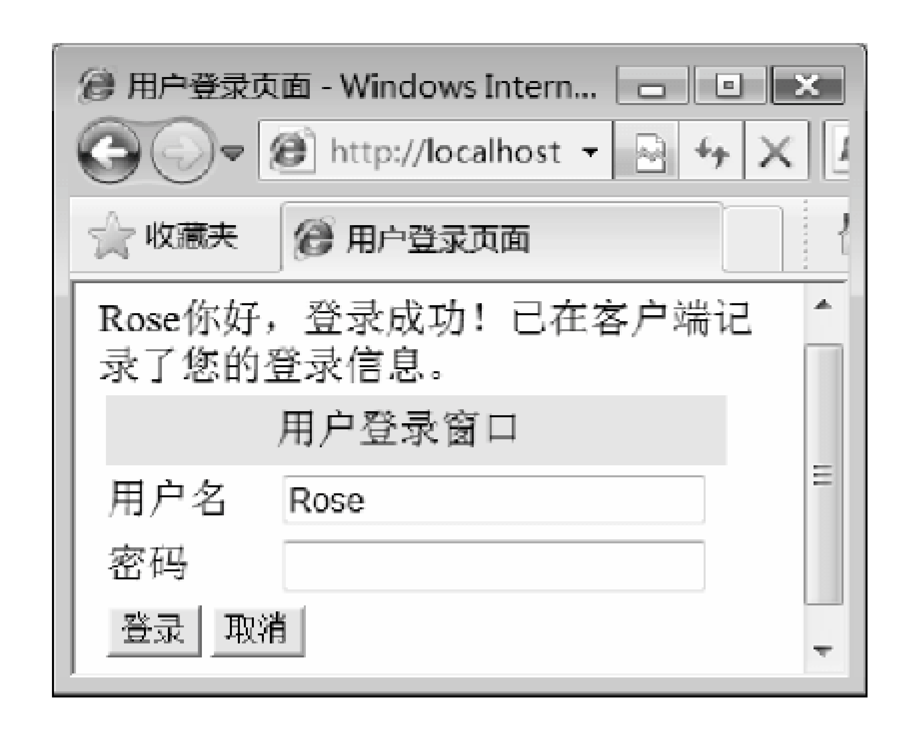


图 3-6　登录提交

当用户再次来访时，显示再次来访信息、访问次数以及用户上次来访的时间，运行结果如图 3-7 所示。

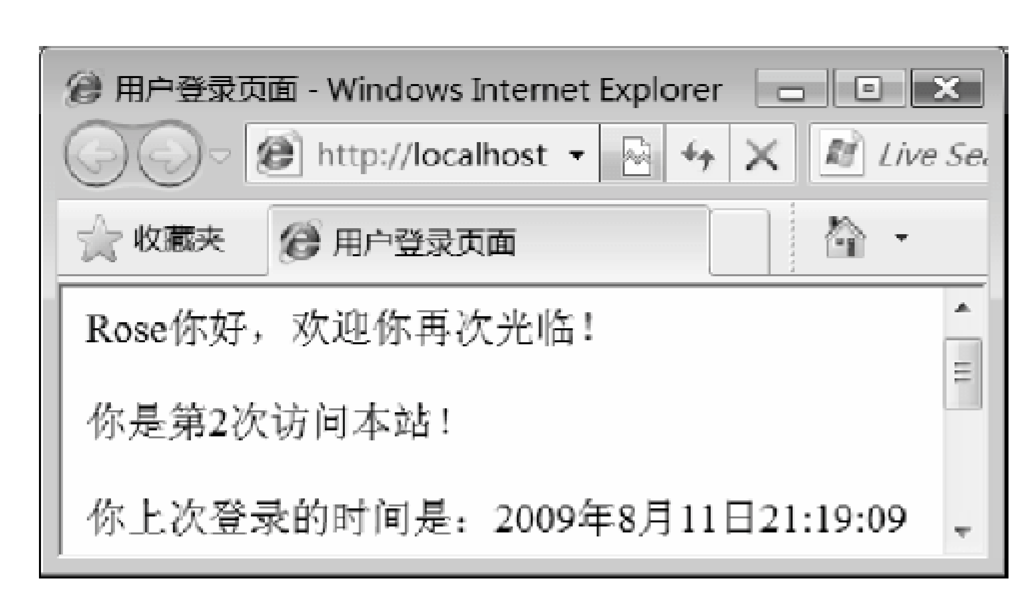


图 3-7　再次访问时

（1）运行 Visual Studio 2005，在“解决方案资源管理器”面板的“在解决方案‘0931’”下新建网站，命名为 Cookies。

(2) 创建用户登录页面 Login.aspx。添加 Label1、Label2 控件和 TextBox1、TextBox2 控件用来输入用户名和密码；添加 Label3、Label4、Label5 控件分别用来显示用户的来访信息、访问次数和上次登录时间；添加 Button1、Button2 按钮用来登录或者取消登录。

Login.aspx 登录页面控件设计可以参照图 3-5。

(3) 为 Login.aspx.cs 的 Page_Load 编写事件代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)  //首次加载页面
    {
        if (Request.Cookies["userName"] != null)
        {
            Label3.Text = Request.Cookies["userName"].Value + "你好,欢迎你再次光
临!" + "<p>";
            if (Request.Cookies["acceNum"] != null)
            {
                //访问次数加 1
                int iNum = Convert.ToInt32(Request.Cookies["acceNum"].Value) + 1;
                Label4.Text = "你是第" + Request.Cookies["acceNum"].Value + "次访
问本站!" + "<p>";
                //将新访问次数写回客户端 Cookie
                HttpCookie acceNumCookie = new HttpCookie("acceNum", iNum.ToString
());
                Response.Cookies.Add(acceNumCookie);
                acceNumCookie.Expires = DateTime.MaxValue;// 设置 Cookie 有效期限
            }
            if (Request.Cookies["acceTime"] != null)
            {
                Label5.Text = "你上次登录的时间是: " + Request.Cookies["
acceTime"].Value + "<p>";
            }
        }
        else
        {
            Response.Write("你好,你是第 1 次光临本站!请登录.<br>");
            //将新访问次数写回客户端 Cookie
            HttpCookie acceNumCookie = new HttpCookie("acceNum", "2");
            Response.Cookies.Add(acceNumCookie);
            acceNumCookie.Expires = DateTime.MaxValue;// 设置 Cookie 有效期限
        }
    }
}
```

(4) 在 Login.aspx 页面的“设计”视图中双击 Button1 按钮，在 Button1_Click 事件中编写事件代码如下。Button1_Click 事件在用户单击 Button1 “登录”按钮的时候执行。

```
protected void Button1_Click(object sender, EventArgs e)
{
```

```
        //将本次登录的日期和时间,写入客户端 Cookie
        HttpCookie acceTimeCookie = new HttpCookie("acceTime", DateTime.Now.ToLongDateString() +
DateTime.Now.ToLongTimeString());
        Response.Cookies.Add(acceTimeCookie);

        //将用户名写入客户端 Cookie
        HttpCookie userNameCookie = new HttpCookie("userName", this.Txt1.Text);
        Response.Cookies.Add(userNameCookie);

        // 设置 Cookie 有效期限
        acceTimeCookie.Expires = DateTime.MaxValue;
        userNameCookie.Expires = DateTime.MaxValue;

        Label3.Text = this.TextBox1.Text + "你好,登录成功!已在客户端记录了您的登录信息.
</br>";
}
```

这里将用户的来访信息用 Label 控件显示，也可以用 Response 对象的 Write 方法把信息直接输出到客户端。例如：

```
Response.Write("你是第" + Request.Cookies["acceNum"].Value + "次访问本站!");
Response.Write("您上次登录的时间是: " + Request.Cookies["acceTime"].Value);
```

3.2.3　小结

（1）删除一个 Cookie 的方法可以是设置一个过期的、同名的 Cookie 来覆盖原来的 Cookie。例如：

```
HttpCookie userNameCookie = new HttpCookie("userName");
userNameCookie.Expires = DateTime.Now.AddDays(-1);
Response.Cookies.Add(userNameCookie);
```

这里为 Cookie 设置了一个一天前的有效期，实际上是让客户端浏览器来删除这个过期的 Cookie。

（2）把 Cookie 信息写到客户端可以有更简单的方法。

语法为：

```
Response.Cookies[Cookie 名称].Value = 变量值;
```

例如：

```
Response.Cookies["userName"].Value = "小松鼠"; //写 Cookie
Response.Cookies["acceNum "].Value = 1;
Response.Cookies["accetime"].Value = DateTime.Now.ToString();
```

（3）注意用户可以通过设置浏览器的方法禁用 Cookie。

3.2.4 思考与练习

如图 3-8 所示,在用户登录窗口实现"记住状态"功能。

提示:在用户选择"记住状态"后将用户登录信息写入客户端 Cookie,当用户再次运行登录页面时,将从客户端读回的 Cookie 信息事先写入输入文本框。

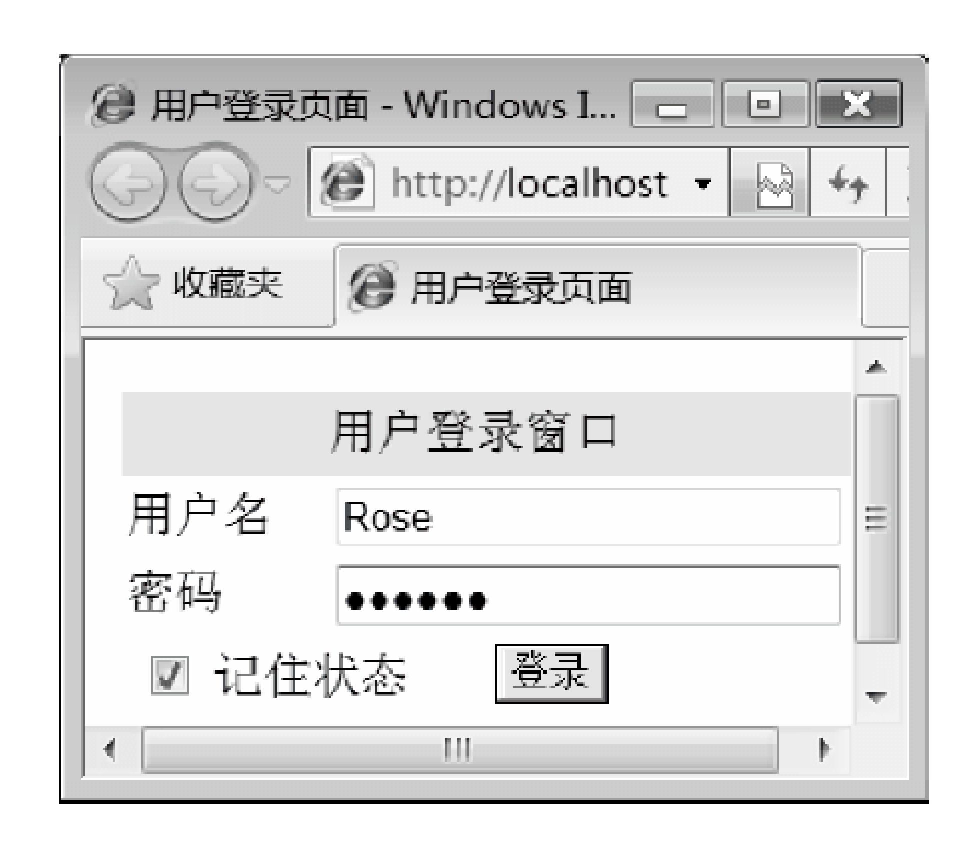


图 3-8 思考与练习

任务 3.3 设计网站聊天室

任务目标

(1) 了解 Session 对象和 Application 对象的属性、方法和事件。

(2) 掌握利用 Session 对象和 Application 对象在页面间保存和传递数据的方法。

3.3.1 Session 对象和 Application 对象简介

Session 对象和 Application 对象实际上可以看成是在一个 Web 应用程序(即一个站点)的不同页面之间保存数据、传递数据的变量。

Session 对象是用来跟踪、面向单个用户的,也即仅供某一个用户使用。Session 对象的 Timeout 属性可以用来设置 Session 的有效时长。例如,Session. Timeout=10;表示在用户停止使用应用程序页面 10 分钟后,该 Session 对象将因超时被清除,默认时长为 20 分钟。

Application 对象是共享的,可以供访问该应用程序的所有用户使用。Application 对象的生命有效期为从应用程序启动到应用程序关闭(Web server 重启)。第一个用户访问站点时创建的 Application 对象全程有效,所有访问本应用程序的用户都可以使用该对象。

Session 对象是 System. Web. HttpSessionState 类的实例;Application 对象是 System. Web. HttpApplicationState 类的实例。

3.3.2　聊天室首页与简单计数器设计

下面通过聊天室实例来说明 Session 对象和 Application 对象的应用。在聊天室设计中，可以使用 Session 对象保存每个用户的昵称、跟踪单个用户的发言；可以使用 Application 对象保存公共发言内容、设计简单计数器统计在线人数等。运行结果如图 3-9 和图 3-10 所示。



图 3-9　聊天室首页

(1) 在 E:\0931 目录下双击 0931.sln 文件，打开 Visual Studio 2005。在“解决方案‘0931’”下新建网站，命名为 Chatroom。默认首页文件为 Default.aspx。

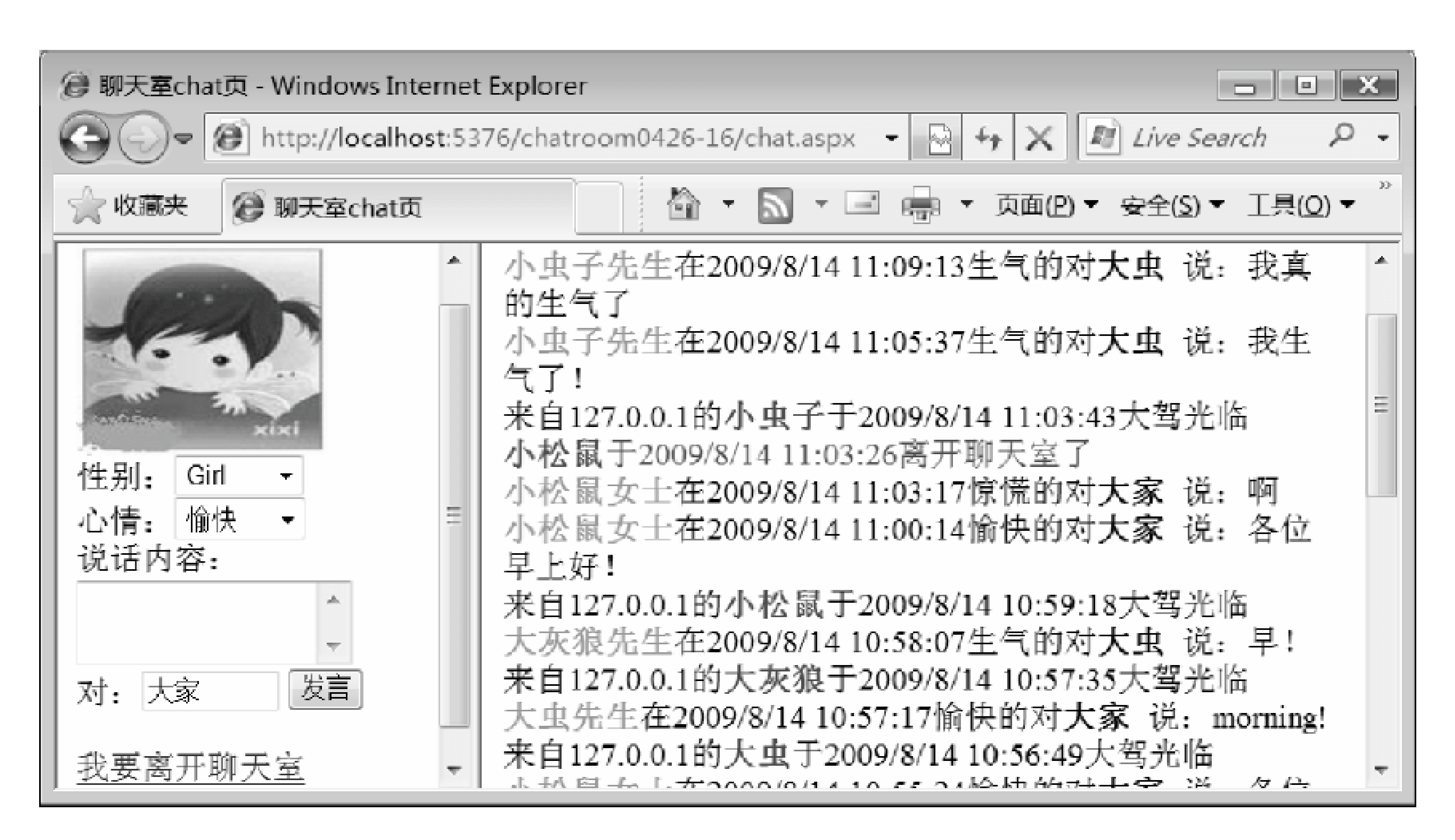


图 3-10　Chat.aspx 页面运行结果

(2) 为 Default.aspx 添加窗体控件，切换到“设计”视图，从左侧工具箱标准组中拖出 2 个 Label 控件、1 个 TextBox 控件、1 个 Button 控件，最后给输入昵称的 TextBox 文本框加必填验证。聊天室首页控件设计可以参见图 3-9。

Default.aspx 实现代码如下：

```
<html xmlns = "http://www.w3.org/1999/xhtml" >
<head runat = "server">
    <title>聊天室首页</title>
</head>
<body>
    <form id = "form1" runat = "server">
    <div>
        <asp:Label ID = "Lab0" runat = "server" Text = "0931 聊天室"></asp:Label>
```

```
        <asp:Label ID = "Lab1" runat = "server" Text = ""></asp:Label>
        <p>请输入昵称 <asp:TextBox ID = "Txt1" runat = "server"></asp:TextBox></p>
        <asp:Button ID = "Btn1" runat = "server" Text = "进入聊天室" OnClick = "Btn1_Click" />
         <asp: RequiredFieldValidator ID = "Requ1"   ControlToValidate = "Txt1" runat =
"server" ErrorMessage = "<br>必须输入昵称!"></asp:RequiredFieldValidator>
    </div>
    </form>
</body>
</html>
```

其中,RequiredFieldValidator 必填验证控件的使用参见任务 4.1。

(3) 在"设计"视图中双击 Btn1 按钮,在 Default. aspx. cs 中编写如下事件代码:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Application["user_online"] = = null)
    {
       Application["user_online"] = 0;
    }
    Application["user_online"] = (int)Application["user_online"] + 1;
    Lab1.Text = "(现在共有" + Application["user_online"].ToString() + "人在线!)";
}
protected void Btn1_Click(object sender, EventArgs e)
{
    if (Page.IsPostBack)       //页面数据回传
    {
       //将用户昵称保存到 Session 对象中
       Session["User_name"] = this.Txt1.Text;
        //重定向到聊天室主页面
       Response.Redirect("chat.aspx");
    }
}
```

说明:

① 在 Page_Load()事件代码中,Application["user_online"] = 0;用来给 Application 对象赋初值。这里用 Application 对象设计了一个简单的站点计数器,统计当前的在线人数,每加载一次本页面,Application["user_online"]加 1。

② 在 Btn1_Click()事件代码中,if (Page.IsPostBack)为 true 时,表示页内有数据提交传送,而非首次加载页面,详细说明可参照任务 3.1。

③ Session["User_name"] = this.Txt1.Text 是把用户输入的昵称保存到一个 Session 对象中。想想为什么不能把昵称保存到一个 Application 对象中?

3.3.3 构建登录字符串与发言字符串

(1) 创建 Chat.aspx 页面文件,页面分框架界面设计参照图 3-10。

Chat. aspx 实现代码如下：

```
<html xmlns = "http://www.w3.org/1999/xhtml" >
<head runat = "server">
    <title>聊天室 Chat.aspx 主页面</title>
</head>
<Frameset cols = "120, * " rows = " * " >
    <Frame name = "say" src = "Inputwin.aspx">
    <Frame name = "message" src = "Showwin.aspx">
</Frameset>
</html>
```

这是一个 HTML 语言编写的分框架页面程序，它把一个窗口分成了两个。左半窗口用来存放输入发言内容的页面文件 Inputwin. aspx，右半窗口用来存放显示聊天内容的页面文件 Showwin. aspx。

(2) 构建登录消息字符串。在 Chat. aspx. cs 的 Page_Load 事件中编写代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
        string user_name = (string)Session["user_name"];     //取用户昵称

        string sayStr = "来自" + (string)Request.ServerVariables["REMOTE_ADDR"] + "的";
        sayStr = sayStr + "<b><font color = red>" + user_name + "</font></b>";
        sayStr = sayStr + "于" + DateTime.Now + "大驾光临" ;

        Application.Lock();
        Application["show"] = sayStr + "<br>" + Application["show"];// I = I + 1
        Application.UnLock();
}
```

这里构建了一个登录消息字符串 sayStr，并且把它放到了一个共享的 Application ["show"]对象中，注意 I=I+1 的写法。Lock()与 UnLock()是 Application 对象的方法，在使用 Application 对象前后分别用来加锁和解锁，以防止出现多个用户同时修改的情况。string user_name=(string)Session["user_name"];是把用户昵称从 Session 对象中取出来。

(3) 构建发言内容字符串。创建输入发言内容的页面文件 Inputwin. aspx。为 Inputwin. aspx 页面添加控件，控件布局可以参见图 3-10 Chat. aspx 页面运行结果界面的左侧窗口。这里用了两个 DropDownList 下拉列表框控件，分别用来选择发言人性别和发言时的心情；一个单行 TextBox 控件(对谁说)；一个多行 TextBox 控件(发言内容)；一个 Button 按钮(发言提交)。最后给输入发言内容的 TextBox 加必填验证控件。

Inputwin. aspx 页面主要代码如下：

```
<form id = "form1" runat = "server">
    <div>
```

```
        <asp:Image ID = "Image1" runat = "server" Height = "99px" ImageUrl = "~/images/girl.jpg"
            Width = "123px" /><br />
    性别:<asp:DropDownList ID = "sex" runat = "server" Width = "63px">
                <asp:ListItem Value = "女士"> Girl </asp:ListItem>
                <asp:ListItem Value = "先生"> Boy </asp:ListItem>
            </asp:DropDownList><br />
    心情:<asp:DropDownList ID = "sayemotional" runat = "server" Width = "64px">
                <asp:ListItem>愉快</asp:ListItem>
                <asp:ListItem>生气</asp:ListItem>
                <asp:ListItem>惊慌</asp:ListItem>
                <asp:ListItem>无表情</asp:ListItem>
            </asp:DropDownList><br />
说话内容:<asp:TextBox ID = "Txt2" runat = "server" Columns = "30" TextMode = "MultiLine"
Height = "25px" Width = "129px"></asp:TextBox><br />
        对:<asp:TextBox ID = "Txt1" runat = "server" Text = "大家" Columns = "10" Width =
"62px"></asp:TextBox>
            <asp:Button ID = "Btn1" runat = "server" Text = "发言" OnClick = "Btn1_Click" />
             <asp:RequiredFieldValidator ID = "Requ1" ControlToValidate = "Txt2" runat =
"server" ErrorMessage = "必须输入发言内容!"></asp:RequiredFieldValidator><br />
            <a href = "exit.aspx" target = "_top">我要离开聊天室</a><br />
    </div>
</form>
```

在“设计”视图中双击 Btn1(发言)按钮，在 Inputwin.aspx.cs 文件的 Btn1_Click 事件中编写代码如下：

```
protected void  Btn1_Click(object sender, EventArgs e)
{
    if (Page.IsPostBack = = true) //有页面数据回传
    {
        String ssex,emotion, who;
        //获取性别
        ssex = sex.SelectedItem.Value;
        //获取发言时心情
        emotion = sayemotional.SelectedItem.Text + "的";
        //获取对谁说
        who = "对" + "<b>" + Txt1.Text + "</b>";
        //构建发言字符串:
        String sayStr = "<font size = '3' color = '00ff00'><b>" + (string)Session["user_name"];
        sayStr = sayStr + ssex + "</b></font>在" + DateTime.Now + emotion + who + " 说:";
        sayStr = sayStr + this.Txt2.Text;
        Application.Lock();
        Application["show"] = sayStr + "<br>" + (string)Application["show"];
        Application.UnLock();
        // 将发言框清空
        this.Txt2.Text = "";
    }
}
```

构建的 sayStr 发言内容字符串也放到了 Application["show"]对象中，即所有登录字

符串和发言字符串是放在同一个 Application["show"]对象中的。

(4) 创建显示发言字符串和发言内容的页面文件(Showwin. aspx),实现代码如下:

```
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<meta http-equiv="refresh" content="10"/>
    <title>显示发言字符串和发言内容的页面</title>
</head>
<body>
    <form id="form1" runat="server">
       <div>
       </div>
    </form>
</body>
</html>
```

这里的<meta http-equiv="refresh"content="10"/>表示每隔 10 秒自动刷新页面一次,用以不断更新页面内容。

在 Showwin. aspx. cs 的 Page_Load 事件中编写代码如下:

```
protected void Page_Load(object sender, EventArgs e)
{
    //在页面上输出 Application["show"]的值
    Response.Write((string)Application["show"]);
}
```

(5) 为离开聊天室页面的 Exit. aspx. cs 文件编写代码如下:

```
protected void Page_Load(object sender, EventArgs e)
{
    //构建离开消息字符串,仍保存在 Application["show"]中
    string sayStr = "<b>" + (string)Session["user_name"] + "</b>";
    sayStr = sayStr + "于" + DateTime.Now + "离开聊天室";
    sayStr = "<font color='green'>" + sayStr + "</font>";
    Application.Lock();
    // I=I+1
    Application["show"] = sayStr +"<br>" +(string)Application["show"];
    //在线人数减 1
    Application["user_online"] = (int)Application["user_online"] -1;
    Application.UnLock();
    Response.Redirect ("Default.aspx");
}
```

(6) 运行聊天室首页文件 Default. aspx。

3.3.4　小结

(1) 聊天室是 Session 对象和 Application 对象应用的典型实例。其他应用的场合还有:用户登录、购物车、站点计数器等。这两个对象用来保持(保存和传递)特定用户或所

有用户的状态信息。

（2）Session 对象和 Application 对象的常用事件有：Session_OnStart、Session_OnEnd 和 Application_OnStart、Application_OnEnd，每一个事件过程都有特定的激活时机，它们通常放在应用程序的 Global.asax 文件中使用，可用来设计一个站点计数器。

3.3.5 思考与练习

1. 修改程序，使得在聊天室发言输入窗口选择性别为 Boy 时，上方的图像随之转换为一个男孩的头像。

*2. ASP.NET 的 Global.asax 文件用来存放 Session 对象和 Application 对象的事件过程。参考.NET 2.0 帮助文档，使用 Global.asax 文件设计实用站点计数器。注意 Global.asax 是应用程序级的文件，必须放在应用程序的根目录下。

本题为选做题（用“*”标注）。

提示：与简单计数器比较，实用计数器需要统计记录访问站点的总人数，并在 Web server 关闭前将总人数保存在一个文本文件中。

第 4章　服务器控件和第三方控件

任务 4.1　验证控件和验证码控件

任务目标

(1) 掌握 ASP.NET 验证控件的作用和使用方法。
(2) 了解常用第三方控件,掌握验证码控件的使用方法。

4.1.1　控件概述

ASP.NET 内置控件有两大类:HTML 服务器控件和 Web 服务器控件。

比如<input id="Text1" type="text" runat="server"/>就表示一个 HTML 服务器控件。它与 HTML 控件的不同之处在于有一个新增的属性 runat="server",只有加了 runat="server"属性,ASP.NET 才会处理它。HTML 服务器控件位于 System. Web. UI. HtmlControls 命名空间中,是由 HtmlControls 类派生的。

而<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>就表示一个 Web 服务器控件。Web 服务器控件提供了统一的事件方法和编程模式,所以在 ASP.NET 的 Web 页面表单设计中,常用的是 Web 服务器控件。Web 服务器控件位于 System. Web. UI. WebControls 命名空间中,是由 WebControls 类派生的。

第三方控件是由微软之外的公司或个人开发的控件,它弥补了 ASP.NET 内置控件的缺陷和不足。网络上提供了很多优秀的、免费的第三方控件,比如 FCKeditor(HTML 在线文本编辑器)、My97DatePicker(JS 版日历控件)、AspNetPager(分页控件)、WebValidates (验证码控件)等。在本书后续章节中,还有大量应用第三方控件的实例。

4.1.2　验证控件与用户注册页面

ASP.NET 2.0 提供了以下 5 种验证控件和 1 个汇总控件。它们是:RequiredField Validator(非空验证控件)、CompareValidator(比较验证控件)、RangeValidator(范围验证控件)、RegularExpressionValidator(正则验证控件)、CustomValidator(用户自定义验证控件)、ValidationSummary(错误信息汇总控件)。

验证控件位于工具箱的"验证" 组中。下面通过一个用户注册页面来说明主要验证控件的使用。

(1) 创建 ASP.NET 应用程序。

在 E:\0931 目录下双击 0931. sln 文件,运行 Visual Studio 2005。在"解决方案资源

管理器"面板中，右击"解决方案'0931'"，在弹出的快捷菜单中选择"添加"→"新建网站"命令，新建 Chap4 站点。

(2) 在站点 Default.aspx 页面上先添加一个 6 行 2 列的表格，并添加控件(包括验证控件)，设计用户注册界面。

Default.aspx 主要代码如下：

```
<form id="form1" runat="server">
    <div>
      <table>
        <tr><td>用户名</td>
            <td><asp:TextBox ID="txtLoginName" runat="server"></asp:TextBox>
            <asp:RequiredFieldValidator ID="rfvLoginName" runat="server" ErrorMessage="请
输入用户名" ControlToValidate="txtLoginName">*</asp:RequiredFieldValidator></td></tr>
        <tr><td>密码</td>
            <td><asp:TextBox ID="txtLoginPwd" runat="server" TextMode="Password">
</asp:TextBox>
            <asp:RequiredFieldValidator ID="rfvLoginPwd" runat="server" ErrorMessage="请
输入密码"  ControlToValidate="txtLoginPwd">*</asp:RequiredFieldValidator></td></tr>
        <tr><td>确认密码</td>
            <td><asp:TextBox ID="txtPwdAgain" runat="server" TextMode="Password">
</asp:TextBox>
              <asp:RequiredFieldValidator ID="rfvPwdAgain" runat="server" ErrorMessage=
"请输入确认密码" ControlToValidate="txtPwdAgain">*</asp:RequiredFieldValidator>
              <asp:CompareValidator ID="cvPwdAgain" runat="server" ErrorMessage="两
次密码不一致" ControlToValidate="txtPwdAgain" ControlToCompare="txtLoginPwd">*</asp:
CompareValidator></td></tr>
          <tr><td>QQ</td>
            <td><asp:TextBox ID="txtQQ" runat="server"></asp:TextBox>
              <asp:RequiredFieldValidator ID="rfvQQ" runat="server" ErrorMessage=
"请输入QQ" ControlToValidate="txtQQ">*</asp:RequiredFieldValidator>
              <asp:RegularExpressionValidator ID="revQQ" runat="server" ErrorMessage="QQ
格式错误" ControlToValidate="txtQQ" ValidationExpression="\d*">*</asp:RegularExpression
Validator></td></tr>
        <tr><td>E_MAIL</td>
            <td><asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
             <asp:RequiredFieldValidator ID="rfvEmail" runat="server" ErrorMessage=
"请输入E_MAIL地址" ControlToValidate="txtEmail">*</asp:RequiredFieldValidator>
            <asp:RegularExpressionValidator ID="revEmail" runat="server" ErrorMessage="E_
MAIL地址格式错误" ControlToValidate="txtEmail" ValidationExpression="\w+([-+.']\w+)*
@\w+([-.]\w+)*\.\w+([-.]\w+)*">*</asp:RegularExpressionValidator></td></tr>
        <tr><td><asp:ValidationSummary ID="vsRegister" runat="server" ShowMessageBox=
"true" ShowSummary="false"/></td>
            <td><asp:Button ID="btnSubmit" runat="server" Text="提交" OnClick=
"btnSubmit_Click1"  /></td></tr>
      </table>
    </div>
</form>
```

说明：

① RequiredFieldValidator 控件对用户名输入进行必填验证。ControlToValidate 属性指定被验证的控件是"txtLoginName"。ErrorMessage 属性显示不能通过验证时的错误信息。

② CompareValidator 控件用来比较控件的值。ControlToValidate 属性指定被验证的控件是"txtPwdAgain"，而 ControlToCompare 属性指定了要与之比较的控件是"txtLoginPwd"。

③ RegularExpressionValidator 控件用来验证输入数据格式是否匹配某种特定的模式（正则表达式）。右击 RegularExpressionValidator 控件，打开 RegularExpressionValidator 控件的属性窗口，单击 ValidationExpression 属性右侧的小矩形按钮，即可弹出“正则表达式编辑器”对话框，选择要使用的正则表达式就可以了，如图 4-1 所示。



图 4-1　“正则表达式编辑器”对话框

④ ValidationSummary 控件用于集中显示来自所有验证控件的错误信息。

4.1.3　使用验证码控件

本书使用的是 WebValidates 验证码控件（在其官方网站免费下载 WebValidates. dll 文件即可）。使用第三方控件的一般步骤如下。

1. 下载.dll 文件并添加到工具箱

在 Visual Studio 2005 的“解决方案资源管理器”面板中，右击站点 Chap4，在弹出的快捷菜单中选择“添加 ASP.NET 文件夹”→Bin 命令。在弹出的对话框中右击“Bin 文件夹”，在弹出的快捷菜单中选择“添加引用”命令，在弹出的“添加引用”对话框中，选择“浏览”选项卡，找到 WebValidates. dll 文件并添加到 Bin 文件夹下。

右击工具箱中的任一控件组，比如这里右击“验证组”，在弹出的快捷菜单中选择“选择项”命令，弹出“选择工具箱项”对话框，选择“.NET Framework 组件”选项卡，如图 4-2 所示。单击“浏览”按钮，在弹出的对话框中选择添加 Bin 文件夹下 WebValidates. dll 文件。

操作完成后，在工具箱验证组中可以看到 SerialNumber 控件，此时该控件就可以像其他 ASP.NET 控件一样正常使用了。

2. 向页面拖放控件并注册

在 Default. aspx 页面增加一行表格行并拖入 SerialNumber 控件。可以看到 Default. aspx 页面上方自动增加了一行控件注册代码：

```
<%@ Register Assembly = "WebValidates" Namespace = "WebValidates" TagPrefix = "cc1"  %>
```

属性 Assembly、Namespace、TagPrefix 分别表示注册的控件名、命名空间、标签前缀。

Default. aspx 页面中新增的表格行以及验证码控件代码如下：

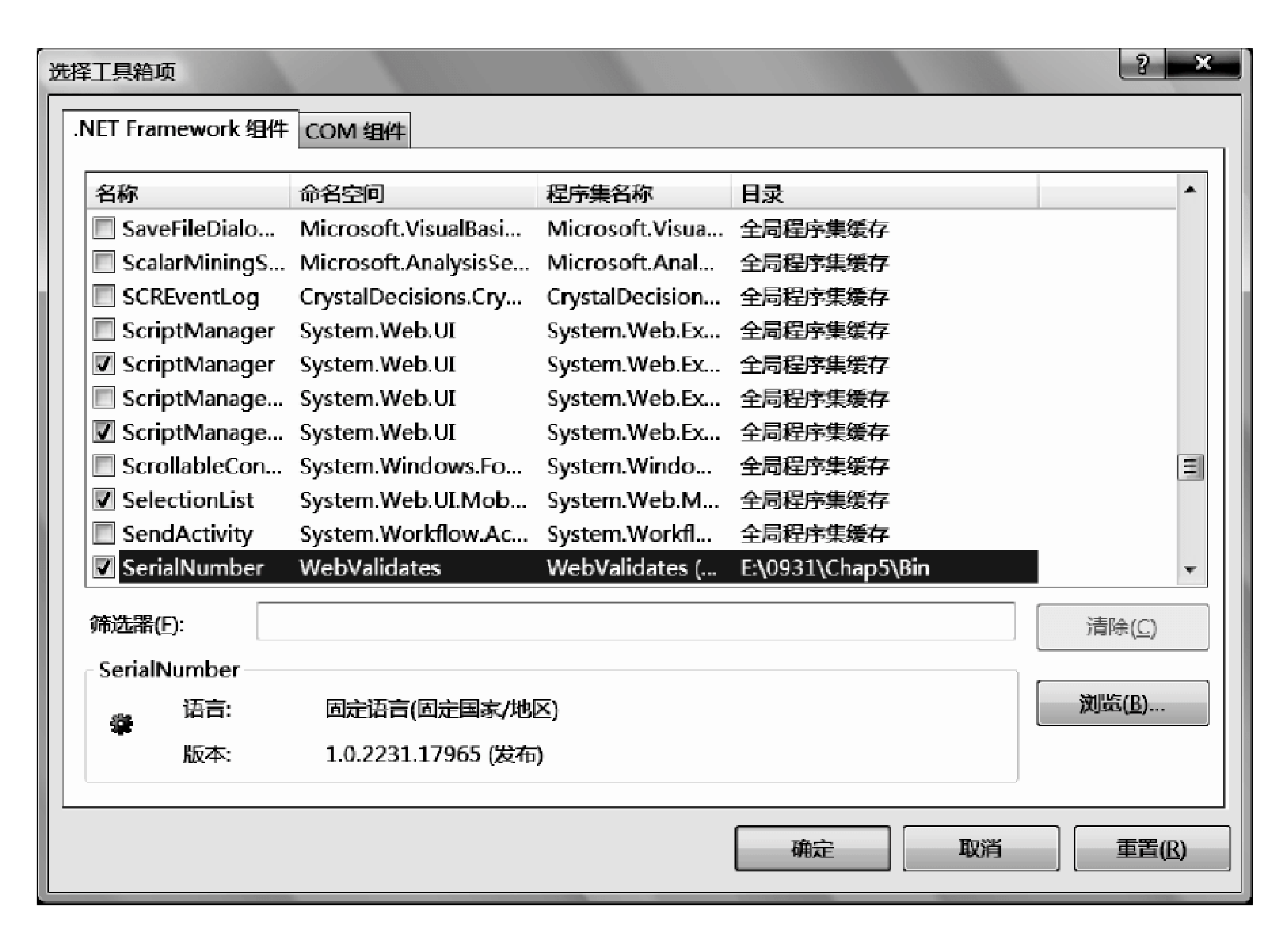


图 4-2 “选择工具箱项”对话框

```
…
<tr>
    <td>验证码</td>
    <td><asp:TextBox ID="txtCode" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="rfvCode" runat="server" ErrorMessage="请输入
验证码" ControlToValidate="txtCode">*</asp:RequiredFieldValidator>
        <cc1:SerialNumber ID="SnCode" runat="server"></cc1:SerialNumber>
        <asp:LinkButton ID="LinkButton1" runat="server" OnClick="LinkButton1_Click1">看不
清换一张</asp:LinkButton></td>
</tr>
```

3. 编写代码生成验证码

在 Default.aspx.cs 文件中编写如下代码:

```
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            SnCode.Create();  //首次加载生成验证码
        }
    }

    protected void btnSubmit_Click(object sender, EventArgs e)
    {
```

```
            if (Page.IsValid)
            {
                if (!CheckCode())
                {
                    Response.Write("<script>alert('验证码错误!')</script>");
                }
            }
        }

        protected void LinkButton1_Click(object sender, EventArgs e)
        {
            SnCode.Create();   //生成新验证码
        }

        private bool CheckCode()
        {
            if (SnCode.CheckSN(txtCode.Text.Trim()))   //判断验证码输入是否正确
            {
                return true;
            }
            else
            {
                SnCode.Create();   //假如验证码输入不正确,则生成新验证码
                return false;
            }
        }
    }
```

运行 Default.aspx,运行效果如图 4-3 和图 4-4 所示。



图 4-3　用户注册页面的验证和验证码控件



图 4-4　来自 Web 页的出错消息

4.1.4　小结

使用 WebValidates 验证码控件时,应注意 SnCode.Create()、SnCode.CheckSN()的

用法。SnCode 是验证码控件的 ID,Create()和 CheckSN()是 WebValidates 验证码控件(类)中自带的方法。代码 SnCode.CheckSN(txtCode.Text.Trim())表示检查比较用户在文本框中输入的验证码,它返回的是一个 bool 类型的值。

4.1.5 思考与练习

为用户登录页面添加输入数据验证功能和验证码功能。

任务 4.2 日历控件和 JS 版日历控件

任务目标

(1) 了解 ASP.NET 日历控件。

(2) 掌握 JS 版日历控件的作用和使用方法。

4.2.1 Calendar 日历控件

使用日历控件可以省去手工输入日期的麻烦。在 ASP.NET 服务器控件中,有一个 Calendar 控件,它使用简便。

在站点 Chap4 下创建一个 Calendar.aspx 页面,并在页面上拖放一个用来输入日期的 TextBox1 控件、一个 Calendar1 日历控件。在页面运行时,当用户在 Calendar1 控件上选择日期后会触发 SelectionChanged 事件。在 Calendar.aspx.cs 文件中编写如下 SelectionChanged 事件代码可将用户选择的日期赋值给 TextBox1 控件。

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    TextBox1.Text = Calendar1.SelectedDate.ToString ();
}
```

Calendar.aspx 运行效果如图 4-5 所示。



图 4-5 Calendar 控件运行效果

4.2.2 JS 版日历控件

My97DatePicker JavaScript(JS)版日历控件是一款功能强大的第三方控件。JS 版日历控件可以从它的官方网站 http://www.my97.net 免费下载。须将下载的整个 My97DatePicker 文件夹放到 Web 站点根目录下才可使用,如图 4-6 所示。

在站点 Chap4 下创建 JSCalendar.aspx 页面文件,在页面中添加一个 TextBox 控件用来输入日期。在 JSCalendar.aspx 中编写如下代码:

```
<html xmlns = "http://www.w3.org/1999/xhtml">
```

```
<head runat = "server"><title>JS 日历控件</title></head>
<body>
    <script language = " javascript " type = " text/javascript " src = " My97DatePicker/
WdatePicker.js"></script>
    <form id = "form1" runat = "server">
       <div>输入日期:
        <asp:TextBox ID = "TextBox1" runat = "server" CssClass = "Wdate" Text = "   "
          onFocus = "new WdatePicker(this,'%Y-%M-%D %h:%m',true,'default')">
          </asp:TextBox>
       </div>
    </form>
  </body>
</html>
```

这里的 onFocus 事件表示当光标移动到 TextBox1 控件上时，就会激活、显示 JS 版日历控件。

运行 JSCalendar.aspx，页面运行效果如图 4-7 所示。

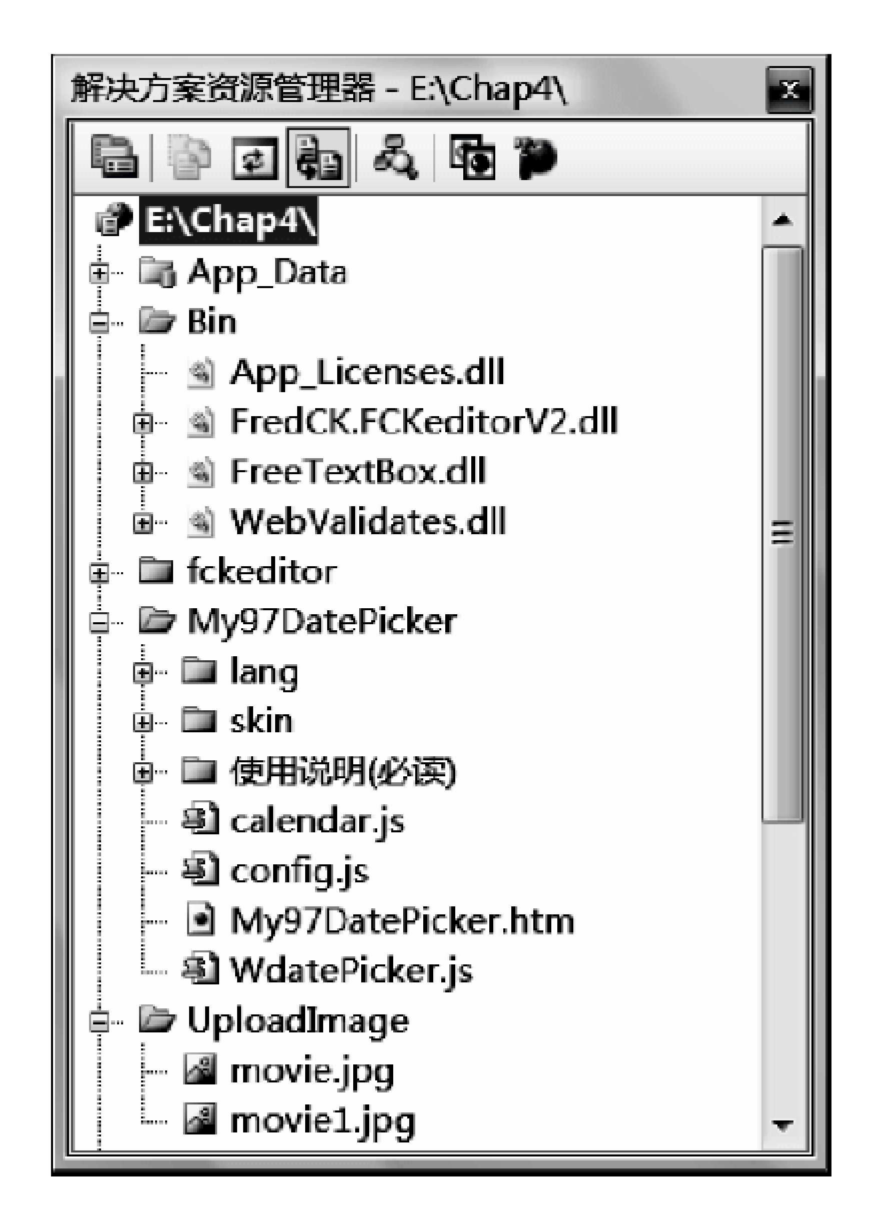


图 4-6　站点 Chap4 目录结构



图 4-7　JS 日历控件运行效果

4.2.3　小结

较之 Calendar 控件，JS 版日历控件设计美观、功能强大，且在用户选择日期时不会造成页面频繁回传(刷新)。

4.2.4　思考与练习

为用户注册页面添加“出生日期”一栏，并使用 JS 日历控件帮助用户输入日期。

任务 4.3 在线文本编辑控件

任务目标

了解在线文本编辑环境 FCKeditor 的使用方法。

4.3.1 下载安装 FCKeditor 控件

FCKeditor 是当前网络上最流行的 HTML 在线文本编辑器，其功能齐全，支持多种开发语言，支持多种浏览器，是一个具有开放接口的国外开源项目。本书在后续的新闻网站和博客日志网站开发中，在输入、编辑新闻内容和日志文章时，将用它对文本格式进行设置、上传新闻图片。

FCKeditor 可在它的官方网站 http://www.fckeditor.net 免费下载。

(1) 下载 FCKeditor_2.6.4.1.zip 和 FCKeditor.Net_2.6.3.zip 压缩包。

(2) 将 FCKeditor_2.6.4.1.zip 解压，可以得到一个以 fckeditor 命名的文件夹，把它放到 Web 层(站点)的根目录下，如图 4-6 所示。

(3) 将 FCKeditor.Net_2.6.3.zip 解压，在其 BIN/debug/2.0 中寻找 dll 文件，将 FredCK.FCKeditorV2.dll 文件添加到工具箱。

在 Visual Studio 2005 的“解决方案资源管理器”面板中，右击站点 Chap4，在弹出的快捷菜单中选择“添加 ASP.NET 文件夹”→Bin 命令。右击“Bin 文件夹”，在弹出的快捷菜单中选择“添加引用”命令，弹出“添加引用”对话框，选择“浏览”选项卡，找到 FredCK.FCKeditorV2.dll 文件并添加到 Bin 文件夹中。

右击工具箱中的任一控件组，比如这里右击“验证组”，在弹出的快捷菜单中选择“选择项”命令，弹出“选择工具箱项”对话框，选择“.NET Framework 组件”选项卡，单击“浏览”按钮，在弹出的对话框中选择添加 Bin 文件夹下的 FredCK.FCKeditorV2.dll 文件。

操作完成后，在工具箱验证组中可以看到 FCKeditor 控件，就可以像其他 ASP.NET 控件一样使用了。

(4) 配置 FCKeditor。

修改 fckeditor\editor\filemanager\connectors\aspx 文件夹下的 config.ascx 文件，将方法 CheckAuthentication()中的“return false” 改为“return true”。

修改 fckeditor 文件夹下 fckconfig.js 文件代码如下：

```
var _FileBrowserLanguage = 'aspx';
var _QuickUploadLanguage = 'aspx';
```

(5) 修改 Web.config 站点配置文件代码如下：

```
<configuration>
  <appSettings>
    <add key = "FCKeditor:BasePath" value = "~/fckeditor/" />
    <add key = "FCKeditor:UserFilesPath" value = "~/UploadImage/" />
```

```
    </appSettings>
<connectionStrings/>
```

其中,第一个应用程序设置名称 BasePath 与值“~/FCKeditor/”指明了 FCKeditor 文件夹的路径;第二个应用程序设置名称 UserFilesPath 与值“~/UploadImage/”则自定义了上传图像的文件夹名称。

也可以在 Visual Studio 2005 菜单栏中选择“网站”→“ASP.NET 配置”命令,在弹出的“ASP.NET 网站管理工具”对话框中选择“应用程序配置”,在随后打开的“应用程序”页面中单击“创建应用程序设置”,在“应用程序设置”下方的文本框中输入相应的名称与值,单击“保存”按钮确认即可。

Web.config 站点配置文件的详细说明参见任务 9.1 和任务 9.2。

4.3.2 在发表文章页面使用 FCKeditor 控件

下面以一个发表日志文章的页面为例说明 FCKeditor 控件的使用方法。

在站点下创建 IssueArticleFCKeditor.aspx 页面文件,在页面上拖放添加 2 个 Label 控件、1 个 TextBox 控件、1 个 FCKeditor 控件。IssueArticleFCKeditor.aspx 页面主要代码如下:

```
<%@ Page Language = "C#" AutoEventWireup = "true" Codebehind = "IssueArticleFckeditor.
aspx.cs"
    Inherits = " IssueArticleFckeditor"  ValidateRequest = "false" %>
<%@ Register Assembly = "FredCK.FCKeditorV2" Namespace = "FredCK.FCKeditorV2" TagPrefix =
"FCKeditorV2" %>
<html xmlns = "http://www.w3.org/1999/xhtml">
<head runat = "server">
    <title>发表日志文章</title>
</head>
<body>
    <form id = "form1" runat = "server">
        <div>
            <asp:Label ID = "Label1" runat = "server" Text = "日志标题"></asp:Label>
            <asp:TextBox ID = "txt_title" runat = "server"></asp:TextBox><br />
            <asp:Label ID = "Label2" runat = "server" Text = "日志正文"></asp:Label><br />
            <FCKeditorV2:FCKeditor ID = "FCKeditor1" runat = "server" BasePath = "~/Fckeditor/"
Height = "280px"></FCKeditorV2:FCKeditor>
            <asp:Button ID = "Button1" runat = "server" Text = "发表文章" OnClick = "Button1_
Click" />
        </div>
    </form>
</body>
</html>
```

运行 IssueArticleFCKeditor.aspx 发表日志页面文件,运行效果如图 4-8 所示。

4.3.3 使用 FCKeditor 控件上传图片

继续运行 IssueArticleFCKeditor.aspx,如图 4-8 所示。单击 FCKeditor 文本编辑器

中的“插入/编辑图像”按钮，打开“图像属性”对话框，选择“上传”选项卡。单击“浏览”按钮，在弹出的对话框中选择要上传的图像文件，单击“发送到服务器上”按钮，最后单击“确定”按钮。系统将在站点根目录下自行创建 UploadImage 文件夹并将上传的图像文件保存于此，并在随后发布文件上传成功的消息，如图 4-9 所示。



图 4-8　在发表日志页面使用 FCKeditor 控件

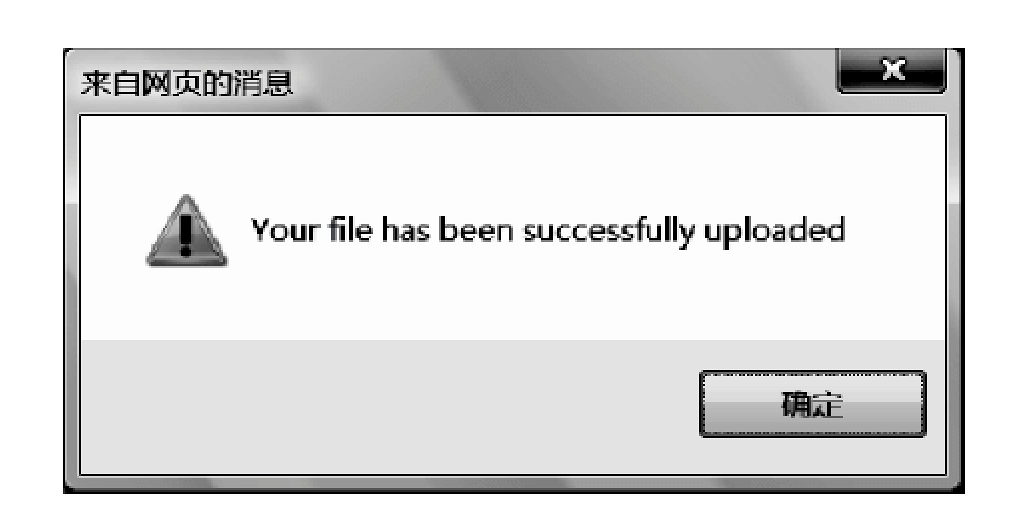


图 4-9　文件上传成功消息

4.3.4　小结

在本任务中介绍了 FCKeditor 控件的基本功能的使用。如需对 FCKeditor 进行进一步应用和讨论，可借助网络和搜索引擎。

ASP.NET 内置控件中有 FileUpload 上传控件，可以上传各种类型的文件至服务器，它的使用简单，读者可以自行上网查询、讨论与尝试。

4.3.5　思考与练习

在 1.2.7 小节思考与练习中的在线留言程序中，为“留言内容”的输入加上在线文本编辑环境。

第 5 章　使用 ADO.NET 访问数据库

任务 5.1　数据库及表的架构和实体类的实现

任务目标

（1）了解三层结构设计模式。

（2）会搭建基于三层结构的系统基本框架。

（3）会利用数据库信息实现实体类。

5.1.1　三层结构概述

在软件体系架构设计中，分层式结构是最常见，也是最重要的一种结构。所谓“分层”，就是将应用程序按照不同的部分划分成不同的模块加以实现，其中每一层实现应用程序一个方面的逻辑功能。采用分层的开发模式，可以对程序员进行合理的分工，提高开发效率，并且编写的应用程序也具有良好的健壮性、可扩展性和便于维护等优点。微软推荐的分层式结构一般分为三层，从下至上分别为：数据访问层、业务逻辑层和表示层。

数据访问层负责对数据库的访问，实现对数据表的增、删、改、查操作。

业务逻辑层负责业务处理和数据传递，它包含了与核心业务相关的逻辑，实现业务规则和业务逻辑。业务逻辑层处于数据访问层与表示层中间，起到了数据交换中承上启下的作用。对于数据访问层而言，它是调用者；对于表示层而言，它却是被调用者。

表示层负责内容的展现和与用户的交互。它位于最外层（最上层），离用户最近，给予用户直接的体验。通俗地讲就是展现给用户的界面。该层主要完成两个任务：①从业务逻辑层获取数据并显示；②与用户进行交互，将相关数据送回业务逻辑层进行处理。

区分层次的目的即为了体现“高内聚，低耦合”的思想。分层的时候如果没有一个适当的数据容器来贯穿各层，将导致耦合性过高，所以用模型层作为在层与层之间数据传递的载体。模型层是标准和规范，它包含了与数据库表相对应的实体类。图 5-1 所示为各层之间的关系。

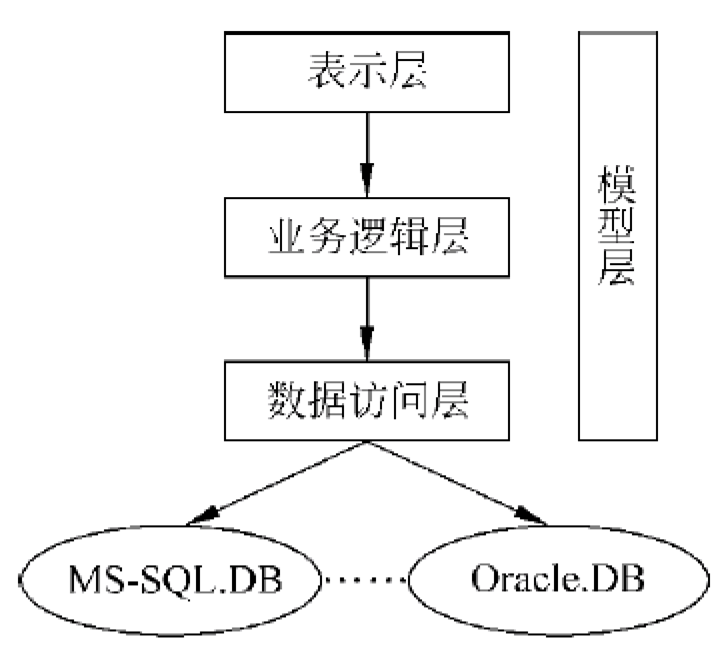


图 5-1　三层的分层式结构

层是一种弱耦合结构，三层之间的依赖是向下的，上层可以使用下层的功能，而下层不能够使用上层的

功能，改变上层的设计对于其调用的下层而言没有任何影响。如单独修改表示层，即对于站点外观的修改不会影响到下面的业务逻辑层和数据访问层。分层设计具有提高应用程序内聚程度、降低应用程序耦合程度、便于应用程序的维护和重用等优点。

5.1.2 搭建基于三层结构的系统基本框架

在简单了解了三层结构的概念之后，下面以开发一个新闻发布系统为例，介绍如何搭建基于三层结构的系统基本框架。根据三层结构设计模式，将整个业务应用划分为模型层、数据访问层、表示层和业务逻辑层，这样有利于系统的开发、维护、部署和扩展。

系统基本框架搭建步骤如下：

(1) 运行 Visual Studio 2005。选择菜单栏“文件”→“新建项目”命令，弹出“新建项目”对话框，在左侧“项目类型”树形目录下选择“其他项目类型”的“Visual Studio 解决方案”，在右侧“模板”列表选择“空白解决方案”选项，新建一个名为 News 的空白解决方案。

(2) 右击该解决方案名，在弹出的快捷菜单中选择“添加”→“新建项目”命令，弹出如图 5-2 所示的“添加新项目”对话框，在左侧“项目类型”树形目录下选择 Visual C#，在右侧“模板”列表选择“类库”选项，新建一个名为 NewsModels 的类库，该类库作为系统的模型层。



图 5-2 新建模型层 NewsModels 界面

(3) 新建名为 NewsDAL 的类库作为系统的数据访问层。操作方法同步骤(2)。

(4) 新建名为 NewsBLL 的类库作为系统的业务逻辑层。操作方法同步骤(2)。

(5) 新建名为 Web 的类库作为系统的表示层。注意此时“添加新项目”对话框的右

侧“模板”列表应该选择“ASP.NET Web 应用程序”选项，如图 5-3 所示。最终生成的解决方案树形目录如图 5-4 所示。

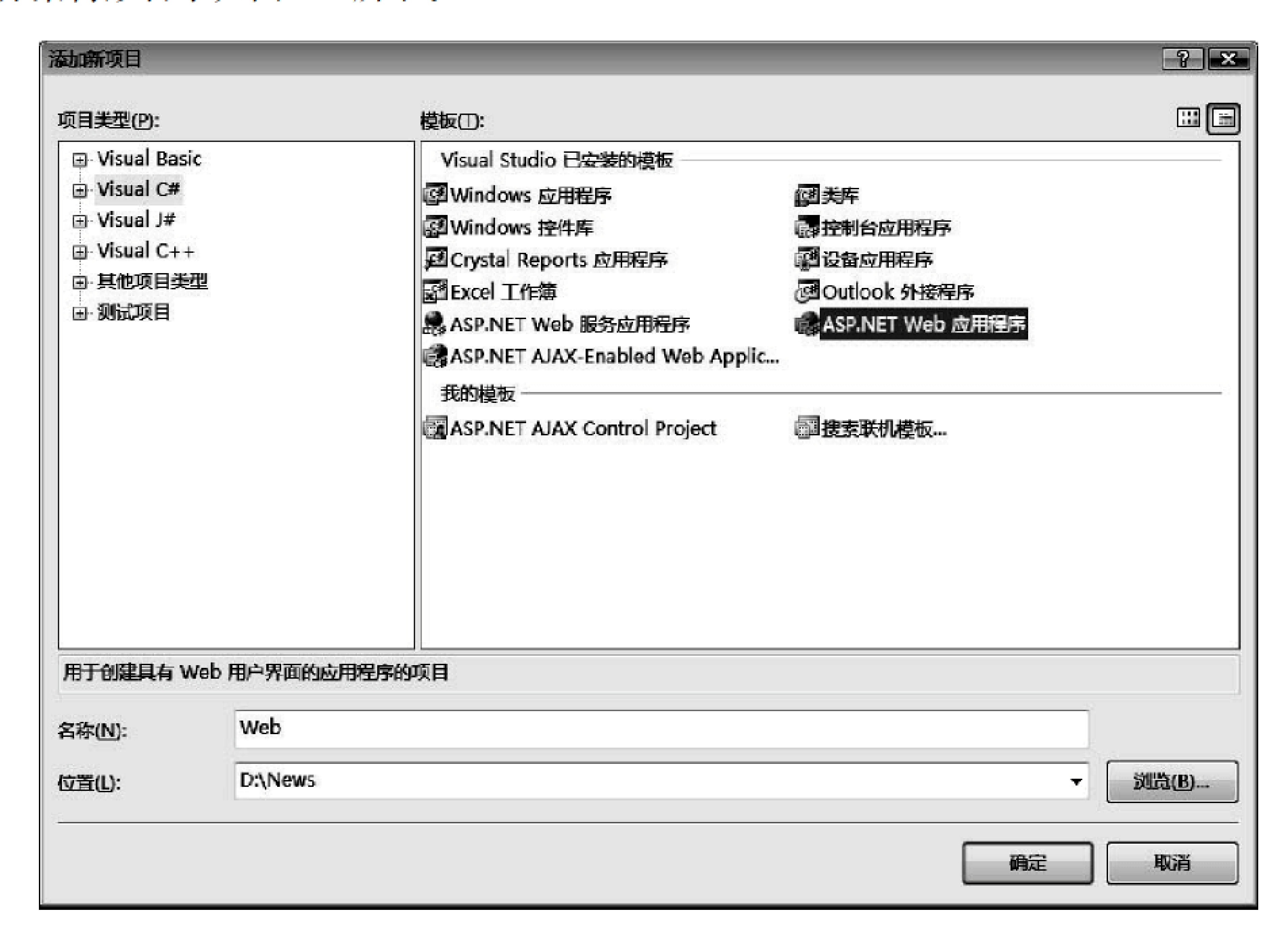


图 5-3　新建表示层 Web 界面

基于三层结构的系统基本框架已经初步搭建成功。但是每一层都是各自独立的，它们之间没有任何联系，因此需要给各层之间建立依赖关系。各层之间相互依赖是它们良好协作的关键。

（1）实现表示层对业务逻辑层的依赖。在“解决方案资源管理器”面板中，单击表示层（Web）前的“＋”号展开树形目录，右击“引用”选项，在弹出的快捷菜单中选择“添加引用”命令，如图 5-5 所示。在弹出的“添加引用”对话框中选择“项目”选项卡，选中项目名称 NewsBLL，单击“确定”按钮，如图 5-6 所示。此时，在表示层的引用目录下就会出现业务逻辑层的项目名称，如图 5-7 所示。



图 5-4　解决方案树形目录



图 5-5　选择“添加引用”命令

（2）实现业务逻辑层对数据访问层的依赖。即在业务逻辑层（NewsBLL）引用数据访问层（NewsDAL）。操作方法同步骤（1）。

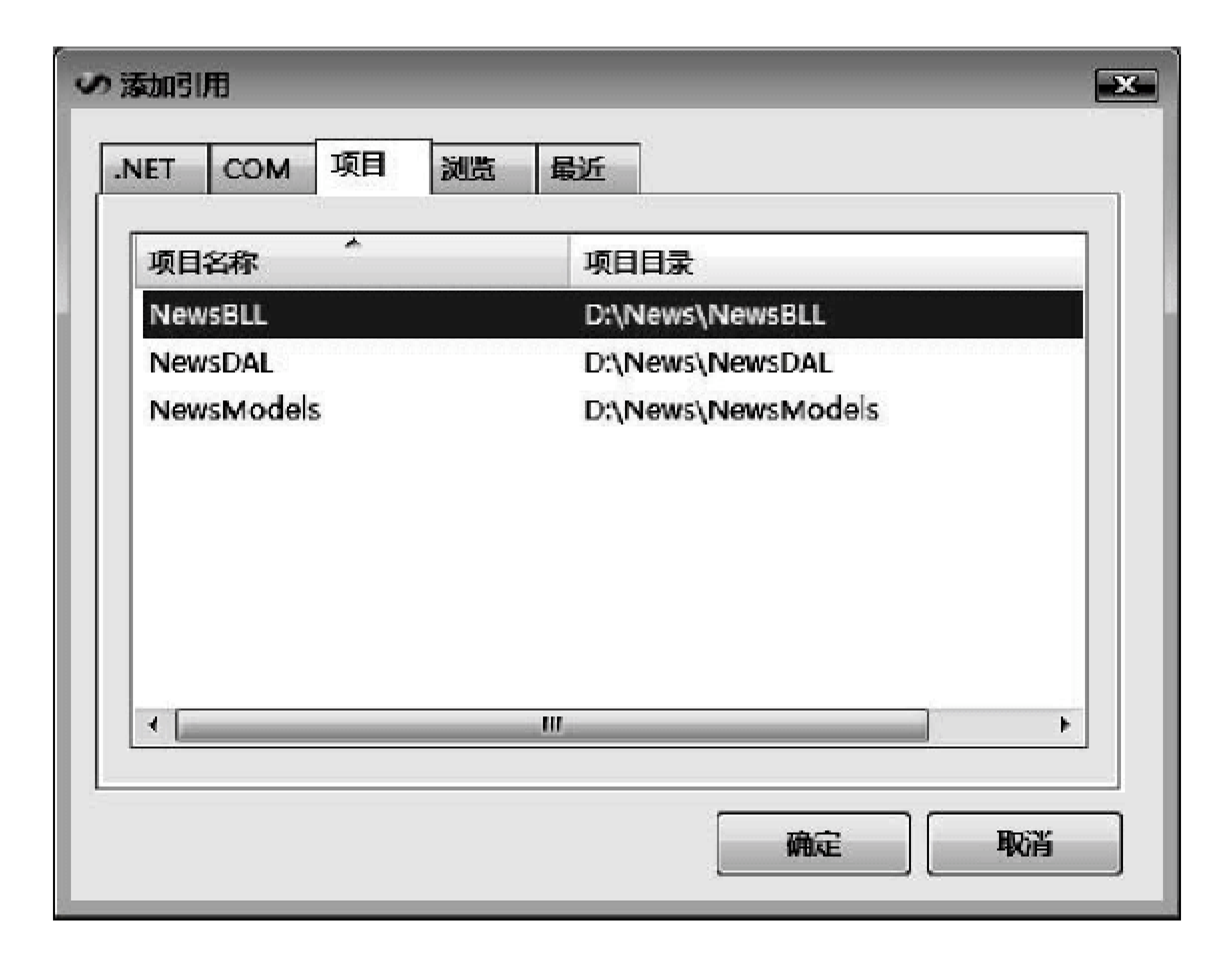


图 5-6 “添加引用”对话框

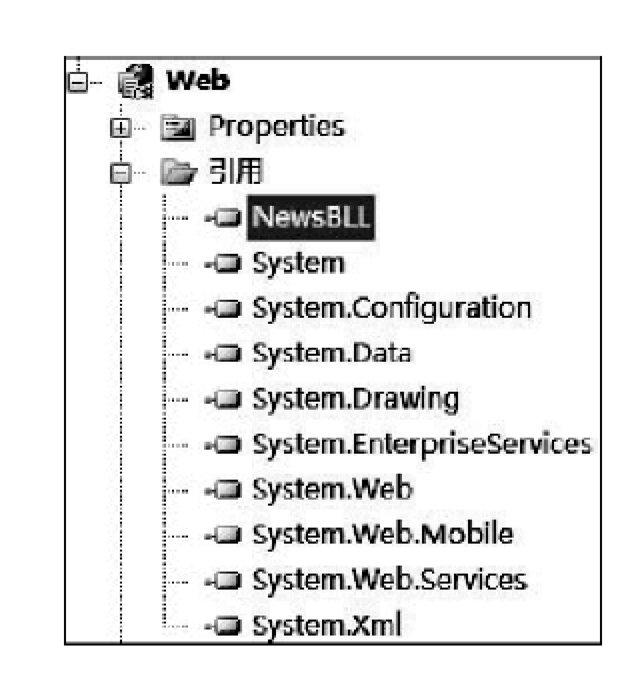


图 5-7 实现引用

(3) 实现数据访问层、业务逻辑层和表示层三层对模型层的依赖。由于数据访问层(NewsDAL)、业务逻辑层(NewsBLL)、表示层(Web)三层之间数据传递的类型是基于模型层的实体类,所以这三层都需添加对模型层(NewsModels)的引用。操作步骤类似,在此不做详细讲解。

至此,基于三层结构的系统基本框架才算真正搭建完成。

5.1.3 分析并创建新闻系统数据库及表的架构

1. 新闻系统功能分析

一个完整的新闻发布系统分前台和后台两个部分:前台主要用于展示新闻、用户留言;后台则用于管理新闻、用户、留言等。

(1) 前台功能

首页模块:导航菜单、站点地图导航、最新新闻列表、最新留言列表

新闻速览模块:新闻分类列表、新闻列表浏览、新闻详细浏览

新闻搜索模块

用户注册模块

用户登录模块

留言板模块

(2) 后台功能

用户管理模块

新闻类别管理模块

新闻文章管理模块

留言管理模块

权限控制模块

2. 数据库设计

根据以上系统功能分析,进行数据库设计,制定用户表、新闻表、新闻类别表、留言表

共 4 张表,每张表的结构信息如表 5-1～表 5-4 所示。

表 5-1　用户表(Users)

字段名称	类　型	说　明
Id	int	用户 ID,主键,自动加 1
LoginName	nvarchar(50)	用户登录名称
LoginPwd	nvarchar(50)	用户登录密码
RealName	nvarchar(50)	用户真实姓名
Address	nvarchar(100)	用户联系地址
Phone	nvarchar(50)	用户联系电话
Email	nvarchar(50)	用户 E-mail 地址
Role	nvarchar(1)	用户角色名称,1 为注册会员,2 为管理员

表 5-2　新闻表(News)

字段名称	类　型	说　明
Id	int	新闻 ID,主键,自动加 1
Title	nvarchar(200)	新闻标题
Author	nvarchar(50)	新闻作者
PubDate	datetime	新闻发表日期
Contents	ntext	新闻内容
Clicks	int	新闻浏览次数
NewsCategoryId	int	新闻类别 ID,外键,关联到 NewsCategories 表中的 ID

表 5-3　新闻类别表(NewsCategories)

字段名称	类　型	说　明
Id	int	新闻类别 ID,主键,自动加 1
Name	nvarchar(50)	新闻类别名称

表 5-4　留言表(Comments)

字段名称	类　型	说　明
Id	int	留言 ID,主键,自动加 1
Title	nvarchar(200)	留言标题
PubDate	datetime	留言发表日期
Contents	ntext	留言内容
UserId	int	留言用户 ID,外键,关联到 Users 表中的 ID

各表之间的关系如图 5-8 所示。

数据库及表的创建步骤如下:

(1) 运行 Visual Studio 2005。在“解决方案资源管理器”面板中,右击项目名 Web,在弹出的快捷菜单中选择“添加”→“新建项”命令,在弹出的“添加新项-Web”对话框中,选择“SQL 数据库”模板,更改名称为 NewsDB.mdf,如图 5-9 所示。

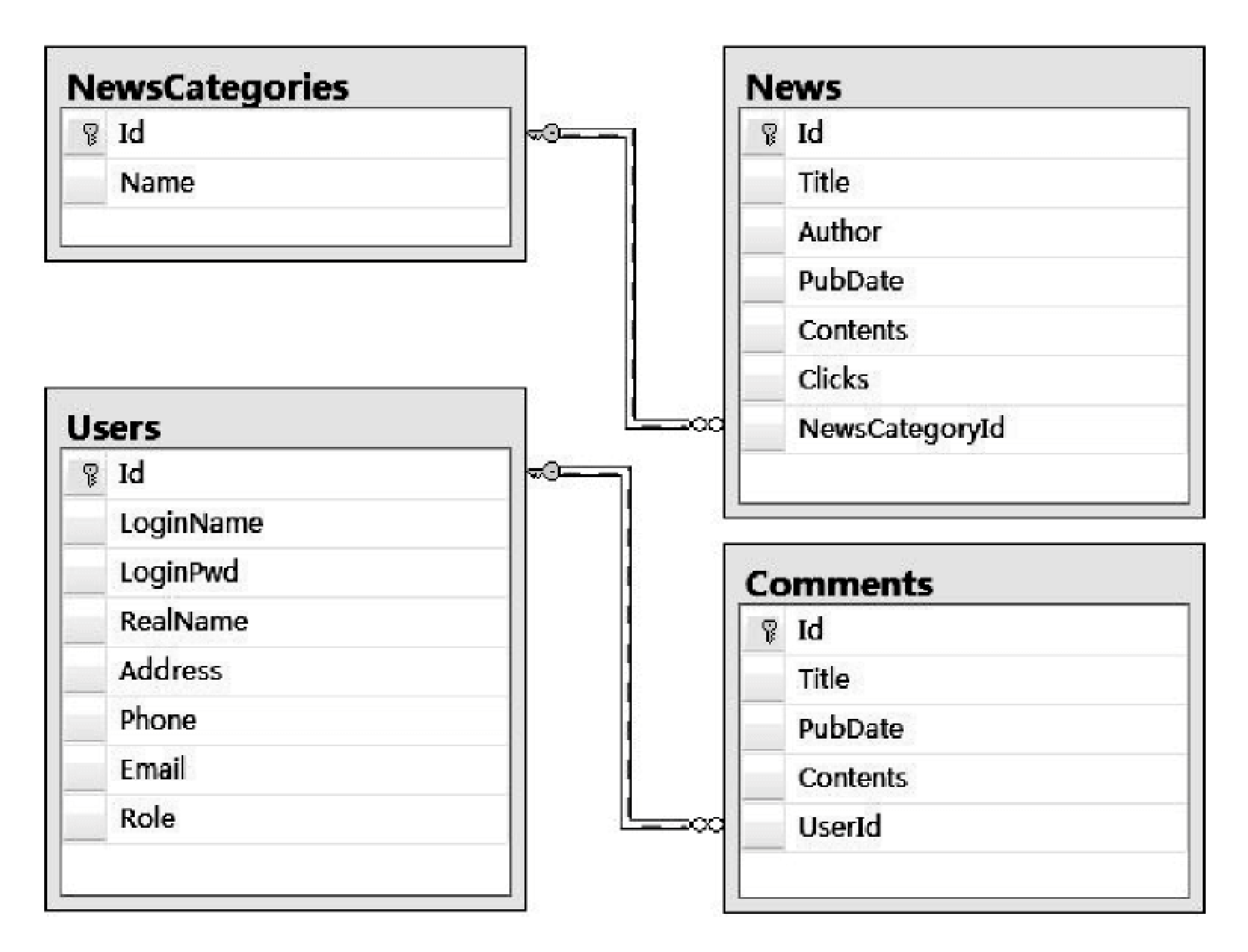


图 5-8　数据库各表之间的关系



图 5-9　新建 NewsDB.mdf 数据库

（2）单击“添加”按钮，弹出如图 5-10 所示的对话框。单击“是”按钮，将数据库 NewsDB.mdf 保存到 App_Data 文件夹中。

（3）在“解决方案资源管理器”面板中，双击数据库名 NewsDB.mdf，弹出“服务器资源管理器”窗口，如图 5-11 所示。

（4）右击“表”目录，在弹出的快捷菜单中选择“添加新表”命令，创建如表 5-1～表 5-4



图 5-10　将数据库放入 App_Data 文件夹中

所示的 4 张表结构。

(5) 右击表名 Users，在弹出的快捷菜单中选择“显示表数据”命令，如图 5-12 所示。在 Users 表中输入几条用户记录。其他各表的数据输入操作类似，在此不再一一讲解。



图 5-11　服务器资源管理器



图 5-12　输入表记录

5.1.4　三层结构系统实体类的实现

实体类就是描述一个业务实体的“类”。整个应用系统业务所涉及的对象(例如新闻系统中的新闻、用户等)都可看成是业务实体，从数据存储的角度来看，业务实体就是存储应用系统信息的数据表。如果将每一个数据表中的字段定义成属性，并将这些属性用一个类封装——这个类就称为“实体类”，如图 5-13 所示。实体类都统一放在模型层中。



图 5-13　User 类与 News 类图

实体类的编写比较简单，通常是根据数据库的字段编写对应的变量和属性，再加一个构造函数即可。下面以 User 类和 News 类为例说明实体类的

创建步骤。

在 NewsModels 项目中分别新建类文件 User.cs 和 News.cs，两个类文件的实现代码如下。

User.cs 的实现代码：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace NewsModels
{
    [Serializable()]
    public class User
    {
        private int id;
        private string loginName = String.Empty;
        private string loginPwd = String.Empty;
        private string realName = String.Empty;
        private string address = String.Empty;
        private string phone = String.Empty;
        private string email = String.Empty;
        private string role = String.Empty;
        public User() { }
        public int Id
        {
            get { return this.id; }
            set { this.id = value; }
        }
        public string LoginName
        {
            get { return this.loginName; }
            set { this.loginName = value; }
        }
        public string LoginPwd
        {
            get { return this.loginPwd; }
            set { this.loginPwd = value; }
        }
        public string RealName
        {
            get { return this.realName; }
            set { this.realName = value; }
        }
        public string Address
        {
            get { return this.address; }
            set { this.address = value; }
        }
        public string Phone
        {
```

```
            get { return this.phone; }
            set { this.phone = value; }
        }
        public string Email
        {
            get { return this.email; }
            set { this.email = value; }
        }
        public string Role
        {
            get { return this.role; }
            set { this.role = value; }
        }
    }
}
```

News.cs 的实现代码：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace NewsModels
{
    [Serializable()]
    public class News
    {
        private int id;
        private string title = String.Empty;
        private string author = String.Empty;
        private DateTime pubDate;
        private string contents = String.Empty;
        private int clicks;
        private int newsCategoryId;
        private NewsCategory newsCategory;
        public News() { }
        public int Id
        {
            get { return this.id; }
            set { this.id = value; }
        }
        public string Title
        {
            get { return this.title; }
            set { this.title = value; }
        }
        public string Author
        {
            get { return this.author; }
            set { this.author = value; }
        }
```

```
            public DateTime PubDate
            {
                get { return this.pubDate; }
                set { this.pubDate = value; }
            }
            public string Contents
            {
                get { return this.contents; }
                set { this.contents = value; }
            }
            public int Clicks
            {
                get { return this.clicks; }
                set { this.clicks = value; }
            }
            public int NewsCategoryId
            {
                get { return this.newsCategoryId; }
                set { this.newsCategoryId = value; }
            }
            public NewsCategory NewsCategory
            {
                get { return this.newsCategory; }
                set { this.newsCategory = value; }
            }
        }
    }
```

这里补充两点说明。为保证数据在不同途径中传递的正确性，将实体类标记为可序列化。另外，数据库中的新闻表(News)和新闻类别表(NewsCategories)之间通过一个外键进行关联，为方便以后程序中的对象访问，在 News 类中构造了一个外键对象属性 NewsCategory。例如，假设有一个新闻实体对象 news，那么使用外键对象后可用 news.NewsCategory.Name 形式获得新闻类别名称。

5.1.5 小结

(1) 通常意义上的三层结构就是将整个业务应用划分为：表示层、业务逻辑层和数据访问层。创建清晰而独立的应用程序层使得应用程序更易于修改。例如，清晰的层次使得无须修改数据访问代码就可以修改用户界面。

(2) 三层结构中各层的依赖顺序是：表示层依赖业务逻辑层；业务逻辑层依赖数据访问层；表示层、业务逻辑层和数据访问层都依赖模型层。

(3) 搭建基于三层结构的系统基本框架的步骤为：搭建模型层→搭建数据访问层→搭建业务逻辑层→搭建表示层→添加各层之间的相互依赖。

(4) 模型层是三层之间数据传递的载体。负责保障对数据库的支持和一致，它包含了与数据库表相对应的实体类。使用实体类更符合面向对象编程的思想，通常把一个表封装成一个类。

5.1.6　思考与练习

编写 NewsCategory 类和 Comment 类代码。

任务 5.2　实现三层结构下的用户登录

任务目标

（1）了解 ADO.NET 的功能和组成。

（2）会用 SqlConnection 对象连接数据库，会用 SqlCommand 对象和 SqlDataReader 对象查询数据。

（3）会实现三层结构下的用户登录功能。

5.2.1　ADO.NET 概述

几乎所有的 Web 应用程序都涉及数据访问，因此，数据访问技术的优劣是 Web 系统成败的关键所在。ADO.NET 是微软公司推出的一种数据访问技术，它是对 ADO（ActiveX Data Objects，.NET 产生之前的数据访问模型）对象模型的扩充，是.NET 应用程序采用的数据访问模型。

ADO.NET 体系结构包括两大核心组件：.NET 数据提供程序和 DataSet 数据集。

.NET 数据提供程序用于连接到数据库、执行命令及检索结果。它包含了许多针对数据源的类库，允许和不同类型的数据源进行交互。对于不同的数据源采用不同的类库，类库名通常是以与之交互的数据源的类型和协议来命名的。表 5-5 列出了一些常见的.NET 数据提供程序，以及它们所使用的 API 前缀和交互的数据源类型。.NET 数据提供程序包含 4 个核心对象，即 Connection 对象、Command 对象、DataReader 对象和 DataAdapter 对象。Connection 对象用于与数据源建立连接；Command 对象用于对数据源执行命令；DataReader 对象用于从数据源中检索只读、只向前数据流；DataAdapter 对象用于将数据源的数据填充 DataSet 数据集并更新数据集。

表 5-5　.NET 数据提供程序

.NET 数据提供程序	API 前缀	数据源描述
ODBC 数据提供程序	Odbc	提供 ODBC 接口的数据源，一般是比较老的数据库 System.Data.Odbc 命名空间
OLE DB 数据提供程序	OleDb	提供 OLE DB 接口的数据源，比如 Access 或 Excel System.Data.OleDb 命名空间
Oracle 数据提供程序	Oracle	Oracle 数据源 System.Data.OracleClient 命名空间
SQL 数据提供程序	Sql	Microsoft SQL Server 数据源 System.Data.SqlClient 命名空间

DataSet 对象表示数据在内存中的缓存，主要用于管理存储在内存中的数据以及对数据的断开操作。

ADO.NET 五大对象之间的关系如图 5-14 所示。

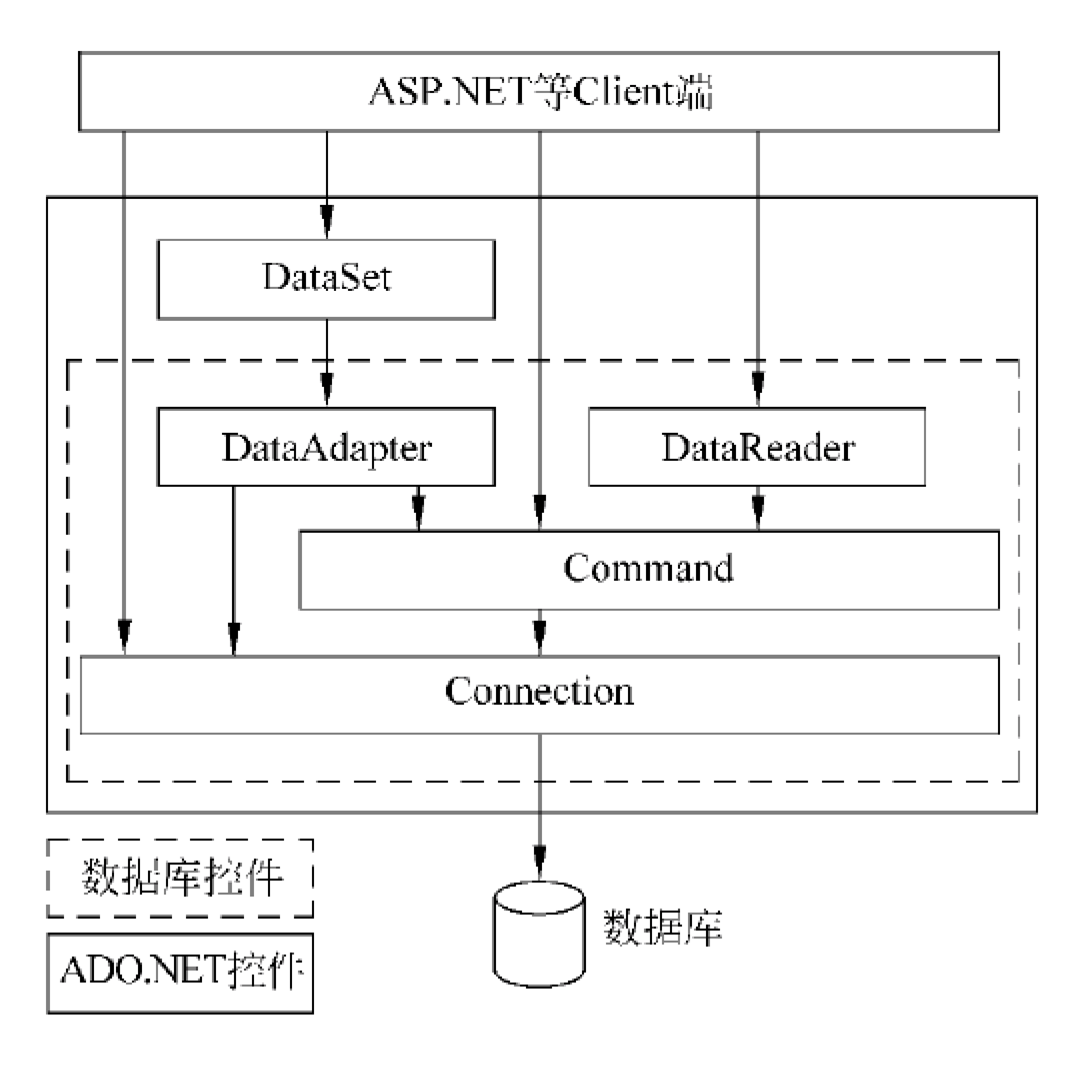


图 5-14　ADO.NET 五大对象之间的关系

举一个实际的例子来理解表 5-5 中的 API 前缀。例如 ADO.NET 中的连接对象（Connection Object），顾名思义，可以通过它建立与数据源的连接。如果使用 OLE DB 数据提供程序连接一个提供 OLE DB 接口的数据源，那么将使用的连接对象就是 OleDbConnection。同理，如果连接 ODBC 数据源或者 SQL Server 数据源就分别加上 Odbc 或者 Sql 前缀，即 OdbcConnection 或 SqlConnection。本书实例使用的数据源是 SQL Server，所以所有的 API 对象都带有 Sql 前缀，比如 SqlConnection。

5.2.2　常用 ADO.NET 对象的使用

下面介绍 ADO.NET 中 SQL 数据提供程序的 4 个核心对象（SqlConnection、SqlCommand、SqlDataReader 和 SqlDataAdapter）和 DataSet 对象的具体使用方法。使用这些对象时，通常在程序代码开头部分用 using 指令引用相应的命名空间。SqlConnection、SqlCommand、SqlDataReader 和 SqlDataAdapter 这 4 个对象用 using System.Data.SqlClient，DataSet 对象用 using System.Data。

1. SqlConnection 对象

在对 SQL Server 数据库中的数据进行操作前首先要连接数据库，连接数据库并完成对数据库的操作之后，必须关闭与数据库的连接。可以使用 SqlConnection 对象来完成。SqlConnection 对象的常用属性与方法如表 5-6 所示。

使用 SqlConnection 对象连接到 SQL Server 数据库的基本步骤如下。

（1）创建一个 SqlConnection 类的对象实例。

（2）设置 SqlConnection 对象的连接字符串属性 ConnectionString。

表 5-6　SqlConnection 对象的常用属性与方法

属　　性	说　　明
ConnectionString	获取或设置用于打开 SQL Server 数据库的连接字符串
方　　法	**说　　明**
Open()	打开数据库连接
Close()	关闭数据库连接

(3) 使用 SqlConnection 对象的 Open()方法或 Close()方法打开或关闭连接。

以连接新闻系统的数据库 NewsDB.mdf 为例，编写 SqlConnection 对象使用代码如下：

```
using System.Data.SqlClient;
…
//建立 SQL 数据库连接对象实例
SqlConnection cn = new SqlConnection();
//设置数据库连接字符串
string connectionString = "Data Source = .\\SQLEXPRESS;AttachDbFilename = |DataDirectory|
\\NewsDB.mdf;Integrated Security = True;User Instance = True";
cn.ConnectionString = connectionString;
//打开数据库连接
cn.Open();
…
//关闭数据库连接
cn.Close();
```

由于数据库连接字符串需要在应用程序多处重复使用，出于对系统以后的可移植、可扩展、可维护性等考虑，一般将连接字符串写在 Web.config 配置文件里。以新闻系统为例，在 Web.config 配置文件里设置如下代码：

```
<connectionStrings>
    <add name = "NewsDBConnectionString" connectionString = "Data Source = .\SQLEXPRESS;
AttachDbFilename = |DataDirectory|\NewsDB.mdf;Integrated Security = True;User Instance =
True" providerName = "System.Data.SqlClient" />
</connectionStrings>
```

这时，在页面中使用该连接字符串，就可以写成：

```
using System.Configuration;
…
string connectionString = ConfigurationManager.ConnectionStrings["NewsDBConnectionString"].
ConnectionString;
SqlConnection 类的构造函数还有一种重载形式：public SqlConnection(string connectionString);
```

所以，对数据库连接对象的操作，还可写成：

```
SqlConnection cn = new SqlConnection(connectionString);
cn.Open();
…
```

```
cn.Close();
```

为了能自动释放数据库连接对象,可使用 using 语句:

```
using (SqlConnection cn = new SqlConnection(connectionString))
{
    cn.Open();
    …
}
```

如上面的代码所示,在 using 语句范围内定义了一个数据库连接对象,当程序执行到 using 语句末尾处时,将自动释放此数据库连接对象,不用再手动写 Close()或 Dispose()方法来关闭或释放资源。用 using 语句,可在程序开发过程中简化代码量,并在一定程度上提高了资源使用效率。

2. SqlCommand 对象

SqlCommand 对象用来对 SQL Server 数据库执行 SQL 命令,如增加、删除、修改、查询等操作。可以使用 SQL 语句,也可以使用存储过程来完成这些操作。SqlCommand 对象的常用属性与方法如表 5-7 所示。

表 5-7 SqlCommand 对象的常用属性与方法

属　性	说　明
Connection	获取或设置 SqlCommand 对象实例使用的数据库连接
CommandText	获取或设置要对数据源执行的 SQL 命令
CommandType	获取或设置 SQL 命令的类别,有 StoredProcedure、TableDirect 或 Text,默认值是 Text
Parameters	获取 SqlParameterCollection 对象
方　法	**说　明**
ExecuteNonQuery()	执行一条不返回结果集的语句,如 INSERT、UPDATE、DELETE 或数据定义语句
ExecuteReader()	将 CommandText 发送到 SqlConnection 并返回 SqlDataReader 对象
ExecuteScalar()	执行查询,并返回查询结果集中第一行的第一列(只有一个值),如执行 COUNT(*)

使用 SqlCommand 对象执行 SQL 命令的基本步骤如下。

(1) 创建一个 SqlCommand 类的对象实例。

(2) 给 SqlCommand 对象的 CommandText 属性设置要执行的 SQL 命令。(如果 SQL 命令中使用了参数,可通过 Parameters 属性为参数设置值。)

(3) 用 SqlCommand 对象的 ExecuteReader()、ExecuteScalar()或 ExecuteNonQuery()等方法执行 SQL 命令。

以实现新闻系统中根据登录名查询用户为例,编写 SqlCommand 对象使用代码如下:

```
using System.Data.SqlClient;
…
```

```
//建立 SQL 命令对象实例
SqlCommand cm = new SqlCommand();
//建立 SQL 命令对象与 SQL 数据库连接对象的连接
cm.Connection = cn;
//设置要执行的 SQL 命令
string sql = "SELECT * FROM Users WHERE LoginName = @LoginName";
cm.CommandText = sql;
//向 SQL 命令对象的参数集添加参数
cm.Parameters.AddWithValue("@LoginName", loginName);
//调用 SQL 命令对象的相应方法执行命令
SqlDataReader dr = cm.ExecuteReader();
…
```

3. SqlDataReader 对象

SqlDataReader 对象用来从 SQL Server 数据库中读取数据，实现对数据源的只读访问。它是面向连接的、只向前的、游标样式的类。由于 SqlDataReader 不需要以随机的方式(即前后移动或根据索引访问)访问数据库，所以不会增加系统的额外开销。与 DataSet 相比，如果执行纯粹的读操作，SqlDataReader 的速度要快许多。SqlDataReader 对象的常用属性与方法如表 5-8 所示。

表 5-8　SqlDataReader 对象的常用属性与方法

属　　性	说　　明
FieldCount	获取当前行中的列数
方　　法	**说　　明**
Read()	读取下一条记录
GetName()	获取指定列的名称
Close()	关闭 SqlDataReader 对象

SqlCommand 对象的 ExecuteReader()方法执行后会返回一个 SqlDataReader 对象，即 SqlDataReader 对象是通过 SqlCommand 对象的 ExecuteReader()方法从数据源中检索行创建的。使用 SqlDataReader 对象执行数据读取的基本步骤如下。

(1) 通过执行 SqlCommand 对象的 ExecuteReader()方法，创建一个 SqlDataReader 类的对象实例。

(2) 用 SqlDataReader 对象的 Read()方法读取数据。(若有多条记录，可通过 while 循环读取。)

(3) 用 SqlDataReader 对象的 Close()方法关闭该对象。

以实现新闻系统中根据登录名查询用户为例，编写 SqlDataReader 对象使用代码如下：

```
using System.Data.SqlClient;
…
//通过执行 SqlCommand 对象的 ExecuteReader()方法来建立 SQL 数据读取对象实例
SqlDataReader dr = cm.ExecuteReader();
//调用 SQL 数据读取对象的 Read()方法读取数据
```

```
if (dr.Read())
{
    …
    //关闭 SQL 数据读取对象
    dr.Close();
    …
}
else
{
    dr.Close();
    …
}
```

4. SqlDataAdapter 对象

SqlDataAdapter 对象表示用于填充 DataSet 和更新 SQL Server 数据库的一组数据命令和一个数据库连接。SqlDataAdapter 对象一般和 DataSet 一起使用,作为 DataSet 与 SQL Server 数据源之间的桥梁来检索和保存数据。SqlDataAdapter 里包含了 SqlConnection 对象,当对数据源进行读取或者写入的时候,SqlDataAdapter 会自动打开或者关闭连接。此外,SqlDataAdapter 还包含对数据进行 SELECT、INSERT、DELETE 和 UPDATE 操作的 SqlCommand 对象的引用。SqlDataAdapter 对象的常用属性与方法如表 5-9 所示。

表 5-9　SqlDataAdapter 对象的常用属性与方法

属　　性	说　　明
SelectCommand	获取或设置一个 SQL 命令,用于在数据源中选择记录
InsertCommand	获取或设置一个 SQL 命令,用于在数据源中插入新记录
DeleteCommand	获取或设置一个 SQL 命令,用于从数据源中删除记录
UpdateCommand	获取或设置一个 SQL 命令,用于更新数据源中的记录
方　　法	**说　　明**
Fill()	填充 DataSet 或 DataTable
Update()	为 DataSet 中每个已插入、已更新或已删除的行调用相应的 INSERT、UPDATE 或 DELETE 语句

5. DataSet 对象

DataSet 对象表示包括相关表、主外键约束和表间关系在内的整个数据集。可将它看成存储数据的数据库,里面包含有一个或多个数据表、数据行、列、约束、表间关系及数据。实际上,它并不是一个真正的数据库,它只是内存中的一个数据集合。数据集是断开式的数据容器,数据集中所有记录可以随机访问。

注意:由于 DataSet 对象能被所有.NET 数据提供程序使用,它不需要指定前缀。

下面以使用 SqlDataAdapter 对象和 DataSet 对象执行数据读取的操作为例,列出基本操作步骤如下。

(1) 用 SqlCommand 对象设置查询命令。

(2) 创建一个 SqlDataAdapter 类的对象实例。

(3) 给 SqlDataAdapter 对象的 SelectCommand 属性设置 SqlCommand 对象。

(4) 创建一个 DataSet 类的对象实例，用于接收执行 SQL 命令返回的数据集。

(5) 用 SqlDataAdapter 对象的 Fill()方法填充数据集。

下面的代码实现了使用 SqlDataAdapter 对象从数据库中选择记录，并用选定的行填充 DataSet 的操作。

```
using System.Data;
using System.Data.SqlClient;
…
//建立 SQL 命令对象,并设置查询命令
SqlCommand cm = new SqlCommand();
cm.CommandText = "SELECT * FROM Users";
cm.Connection = cn;
//建立 SQL 数据适配器对象实例
SqlDataAdapter da = new SqlDataAdapter();
//要求 SqlDataAdapter 对象执行查询命令
da.SelectCommand = cm;
//建立数据集对象
DataSet ds = new DataSet();
//向数据集填充数据
da.Fill(ds);
```

总结：ADO.NET 是用于和数据源交互的.NET 技术。它包含了许多数据提供程序，分别用于访问不同的数据源——取决于它们所使用的数据库或者协议。然而无论使用什么样的数据提供程序，与数据源进行交互的对象的使用方法都是相似的。SqlConnection 对象用于管理与数据源的连接。SqlCommand 对象可以向数据源发送 SQL 命令。SqlDataReader 可以快速地从数据源获得只读的、只向前的数据流。使用 DataSet 可以处理那些已经断开的数据（存储在内存中的），并通过 SqlDataAdapter 实现数据源的读取和写入。

在了解并掌握了常用 ADO.NET 对象的使用方法后，可以开始动手编写新闻系统中用户登录模块了。下面分别从数据访问层、业务逻辑层和表示层来实现用户登录功能。

5.2.3　用户登录数据访问层的实现

数据访问层封装了与数据库交互的操作，包括对数据表的增、删、改、查操作。显然，用户登录要做数据查询。这里将用户登录的数据访问层的操作流程描述为：当用户登录系统时，将用户输入的登录名提交数据库进行查询，如果查询的登录名存在，返回一个用户对象，不存在则返回 null。

在 NewsDAL 项目中新建一个类文件 UserService. cs。注意，以后将数据访问层中有关用户对象的操作都写在这个类里。在 UserService 类中，定义一个根据登录名查询用户的方法 public static User GetUserByLoginName(string loginName)。

需要使用的对象及步骤如下：

(1) 实例化 SqlConnection 对象，实现数据库连接。

（2）实例化 SqlCommand 对象，执行 SQL 命令。

（3）实例化 SqlDataReader 对象，读取数据。

（4）使用实体对象传递数据。

具体实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using NewsModels;
namespace NewsDAL
{
    public static class UserService
    {
        private static string connectionString = ConfigurationManager.ConnectionStrings
["NewsDBConnectionString"].ConnectionString;
        /// <summary>
        /// 根据登录名查询用户
        /// </summary>
        /// <param name="loginName">用户登录名</param>
        /// <returns>用户对象</returns>
        public static User GetUserByLoginName(string loginName)
        {
            using (SqlConnection cn = new SqlConnection(connectionString))
            {
                cn.Open();
                SqlCommand cm = new SqlCommand();
                cm.Connection = cn;
                string sql = "SELECT * FROM Users WHERE LoginName = @LoginName";
                cm.CommandText = sql;
                cm.Parameters.AddWithValue("@LoginName", loginName);
                SqlDataReader dr = cm.ExecuteReader();
                if (dr.Read())
                {
                    User user = new User();
                    user.Id = (int)dr["Id"];
                    user.LoginName = (string)dr["LoginName"];
                    user.LoginPwd = (string)dr["LoginPwd"];
                    user.RealName = (string)dr["RealName"];
                    user.Address = (string)dr["Address"];
                    user.Phone = (string)dr["Phone"];
                    user.Email = (string)dr["Email"];
                    user.Role = (string)dr["Role"];
                    dr.Close();
                    return user;
                }
                else
```

```
                {
                    dr.Close();
                    return null;
                }
            }
        }
    }
}
```

上面的 if 语句里，先构造一个 user 对象，其属性值依次被赋值为数据库中查找到的某条用户记录的各个字段值，然后将该 user 对象返回。

5.2.4　用户登录业务逻辑层的实现

业务逻辑层应该提供哪些方法一般根据实际需求来确定。例如，当用户登录系统时，首先要判断输入的登录信息是否有效。可以在业务逻辑层创建一个对应的用户登录方法，接收表示层提交过来的登录名和密码，判断是否为合法用户。如果是合法用户，返回 true；否则返回 false。

在 NewsBLL 项目中新建一个类文件 UserManager.cs。类似数据访问层中 UserService 类的做法，以后将业务逻辑层中有关用户对象的操作都写在 UserManager 类里。在 UserManager 中，定义一个验证用户登录信息是否有效的方法 public static bool UserLogin (string loginName，string loginPwd，out User validUser)。具体实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
using NewsModels;
using NewsDAL;
namespace NewsBLL
{
    public static class UserManager
    {
        /// <summary>
        /// 登录验证(判断用户登录名是否存在,密码是否正确,验证是否为合法用户)
        /// </summary>
        /// <param name="loginName">用户登录名</param>
        /// <param name="loginPwd">用户密码</param>
        /// <param name="validUser">用户对象</param>
        /// <returns>布尔值</returns>
        public static bool UserLogin(string loginName, string loginPwd, out User validUser)
        {
            User user = UserService.GetUserByLoginName(loginName);
            if (user == null)                //登录名不存在
            {
                validUser = null;
                return false;
```

```
                    }
                    else if (user.LoginPwd != loginPwd) //密码错误
                    {
                        validUser = null;
                        return false;
                    }
                    else
                    {
                        validUser = user;
                        return true;
                    }
                }
            }
        }
```

注意：这个方法的形参 validuser 前使用了 out 关键字，当某个方法需要有多个返回值时，可用它来传递返回值。out 在这里的作用是返回一个用户对象，即当用户合法时，将该用户对象返回，以备在表示层中调用，比如可将该用户对象的相关信息存入 Session 或 Cookie。

5.2.5 用户登录表示层的实现

表示层用于显示数据和接收用户输入的数据，为用户提供一种交互式操作的界面。在表示层中，如果仅仅用于展示内容，可能只需要将控件绑定数据即可，不需要编写代码；如果需要和用户交互，就要编写相关的事件代码。例如，新闻系统中的用户角色分注册会员和管理员两类，以管理员登录页为例，用户单击“登录”按钮的事件，就需要进行用户输入内容的非空验证，然后通过调用业务逻辑层的相关方法判断用户名和密码是否匹配，编写代码验证用户的身份是否为管理员等。

下面在表示层为系统创建管理员登录页。先在 Web 项目中新建一个 Admin 文件夹，在该文件夹下创建页面文件 AdminLogin.aspx。

注意：以后将有关后台管理员操作的页面都统一做在 Admin 文件夹下，系统的样式文件、外观文件、图片文件等都统一放在主题文件夹 App_Themes\Default 下，而系统的一些公用自定义类则都放在 Common 文件夹下，如图 5-15 所示。这样做能使系统文件架构更加清晰，便于系统的管理和维护。

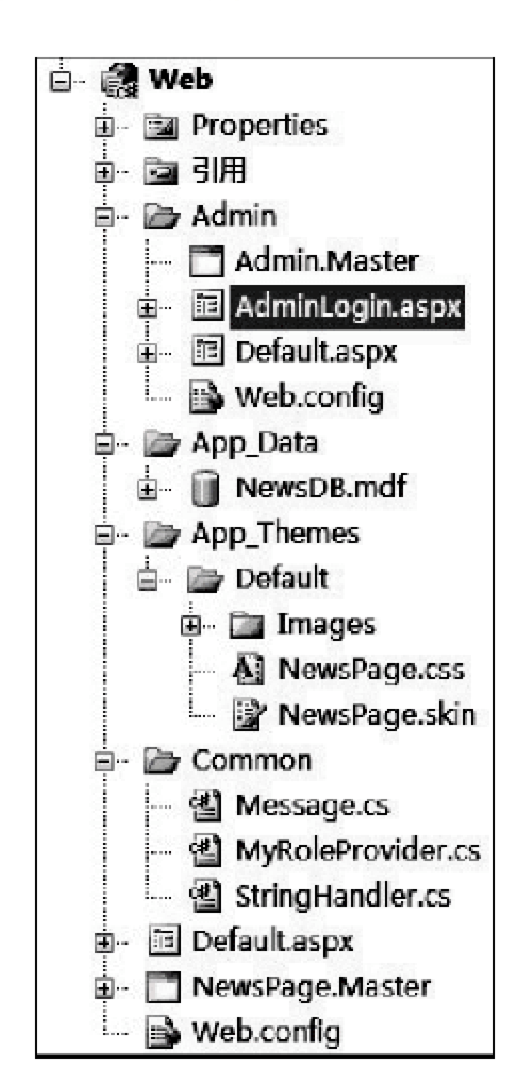


图 5-15 管理员登录页所在文件夹

AdminLogin.aspx 主要实现代码如下：

```
<form id="form1" runat="server">
<div>
  <table>
    <tr><td>用户名: </td>
      <td><asp:TextBox runat="server" ID="txtLoginName"></asp:TextBox>
```

```
                    <asp:RequiredFieldValidator ID = "rfvLoginName" runat = "server" ErrorMessage =
"*"ControlToValidate = "txtLoginName"></asp:RequiredFieldValidator></td></tr>
        <tr><td>密码: </td>
        <td><asp:TextBox runat = "server" ID = "txtLoginPwd" TextMode = "Password"></asp:
TextBox>
                    <asp:RequiredFieldValidator ID = "rfvLoginPwd" runat = "server" ErrorMessage =
"*" ControlToValidate = "txtLoginPwd"></asp:RequiredFieldValidator></td></tr>
          <tr><td colspan = "2" class = "content_center"><asp:Button ID = "btnLogin" runat = "
server" Text = "登录" OnClick = "btnLogin_Click" /></td></tr>
    </table>
  </div>
</form>
```

AdminLogin.aspx.cs 实现代码如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using NewsModels;
using NewsBLL;
namespace Web.Admin
{
    public partial class AdminLogin : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
        protected void btnLogin_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                string loginName = txtLoginName.Text.Trim();
                string loginPwd = txtLoginPwd.Text.Trim();
                User user;
                if (UserManager.UserLogin(loginName, loginPwd, out user))//判断用户名和
                                                                          密码是否正确
                {
                    if (user.Role == "2")  //判断是否为管理员身份
                    {
                        //将用户名和角色信息存入 Cookie
```

```
                        System. Web. Security. FormsAuthentication. SetAuthCookie ( user.
LoginName, false);
                        FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(1,
user.LoginName, DateTime.Now, DateTime.Now.AddMinutes(30), false, "admin");
                        string hashticket = FormsAuthentication.Encrypt(ticket);
                        HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsCookie-
Name, hashticket);
                        Response.Cookies.Add(cookie);
                        if (Request["ReturnUrl"] != null)
                        {
                            Response.Redirect(Request["ReturnUrl"]);
                        }
                        else
                        {
                            Response.Redirect("~/Admin/Default.aspx");
                        }
                    }
                    else
                    {
                        Common.Message.RegScript(this, "你不是管理员,无权访问!");
                    }
                }
                else
                {
                    Common.Message.RegScript(this, "用户名或密码不正确,请重新填写!");
                }
            }
        }
    }
}
```

管理员登录页运行效果如图 5-16 所示。页面的样式、外观控制代码参见系统源程序。



图 5-16　管理员登录页运行效果

5.2.6　小结

(1) ADO.NET 包括两大核心组件：.NET 数据提供程序和 DataSet 数据集。其中，前者包含 4 个核心对象，即 Connection 对象、Command 对象、DataReader 对象和 DataAdapter 对象。

(2) using 关键字在 C# 中有两种用途：一种作为指令，引用命名空间；另一种作为语句，用于定义一个范围，在此范围末尾自动释放对象。

(3) 使用 SqlDataReader 对象执行数据读取操作时，先用 SqlCommand 对象的 ExecuteReader()方法创建一个 SqlDataReader 类的对象实例，再用 SqlDataReader 对象的 Read()方法读取数据(若有多条记录，可通过 while 循环读取)，最后用 SqlDataReader 对象的 Close()方法关闭该对象。

5.2.7　思考与练习

1. ADO.NET 包括哪两个主要的组件？它们分别为哪 5 个对象？各有什么作用？
2. 编程实现基于三层结构的注册会员登录功能。

任务 5.3　实现三层结构下的用户注册

任务目标

(1) 会用 SqlCommand 对象的 ExecuteNonQuery()方法添加数据。

(2) 会实现三层结构下的用户注册功能。

5.3.1　用户注册数据访问层的实现

用户注册就是向数据库添加新的用户数据。在数据访问层中，将添加用户的操作定义成方法 public static void AddUser(User user)。

在 NewsDAL 项目的 UserService 类中增加如下代码：

```
/// <summary>
/// 添加用户
/// </summary>
/// <param name="user">用户对象</param>
public static void AddUser(User user)
{
    using (SqlConnection cn = new SqlConnection(connectionString))
    {
        cn.Open();
        SqlCommand cm = new SqlCommand();
        cm.Connection = cn;
        string sql =
        "INSERT Users (LoginName, LoginPwd, RealName, Address, Phone, Email, Role)" +
        "VALUES (@LoginName, @LoginPwd, @RealName, @Address, @Phone, @Email, @Role)";
```

```
            cm.CommandText = sql;
            cm.Parameters.AddWithValue("@LoginName", user.LoginName);
            cm.Parameters.AddWithValue("@LoginPwd", user.LoginPwd);
            cm.Parameters.AddWithValue("@RealName", user.RealName);
            cm.Parameters.AddWithValue("@Address", user.Address);
            cm.Parameters.AddWithValue("@Phone", user.Phone);
            cm.Parameters.AddWithValue("@Email", user.Email);
            cm.Parameters.AddWithValue("@Role", user.Role);
            cm.ExecuteNonQuery();
        }
    }
```

上面代码中，cm.ExecuteNonQuery()执行 INSERT 语句向数据库的 Users 表添加了一条用户记录。而使用参数化的 INSERT 语句可使程序结构更加清晰，提高程序的可读性。

5.3.2 用户注册业务逻辑层的实现

为了保证用户名唯一，在用户注册时，先根据登录名查询该用户是否已经存在，如果用户名不存在，则可以向数据库添加该用户信息；如果用户名已存在，则返回注册失败信息。这里在业务逻辑层创建一个用户注册的方法 public static bool UserRegister (User user)，这个方法和任务 5.2 中 UserManager.UserLogin(string loginName, string loginPwd, out User validUser)相似，都是对业务逻辑的处理。

业务逻辑层除了包含业务逻辑的处理外，常常还用作表示层与数据访问层之间的数据传递。一般数据访问层公开的方法会在业务逻辑层有个相对应的方法。例如，对于数据访问层的 UserService.AddUser(User user)方法，在业务逻辑层还需再创建一个相应的 public static void AddUser(User user)方法。该方法没有业务逻辑上的处理，仅仅是调用了一下数据访问层的相关方法，为表示层提供所需的数据。

在 NewsBLL 项目的 UserManager 类中增加如下代码：

```
public static void AddUser(User user)
{
    return UserService.AddUser(user);
}
/// <summary>
/// 注册新用户
/// </summary>
/// <param name="user">用户对象</param>
/// <returns>布尔值</returns>
public static bool UserRegister(User user)
{
    if (UserService.GetUserByLoginName(user.LoginName) != null)
    {
        return false;
    }
    else
```

```
        {
            AddUser(user);
            return true;
        }
    }
```

其中，UserService.GetUserByLoginName()在任务 5.2 中已经创建。

5.3.3　用户注册表示层的实现

用户注册表示层除了将接收的用户注册信息传递给业务逻辑层处理外，还需要做一系列的注册信息验证，比如各种输入内容的非空验证、两次输入密码的比较验证、电子邮件格式的合法性验证，以及为防止系统被恶意攻击要求用户输入验证码等。前面列出的几种验证方式可以借助 Visual Studio 2005 工具箱中"验证"选项卡提供的几种验证控件来实现。至于验证码功能，通过引入一个第三方控件——WebValidates 验证码控件来实现。

先在 Web 项目中新建一个 Member 文件夹用于存放与注册会员操作相关的页面，然后在该文件夹下创建页面文件 UserRegister.aspx。

UserRegister.aspx 关键代码如下：

```
<table>
    <tr><td>用户名</td>
        <td><asp:TextBox ID = "txtLoginName" runat = "server"></asp:TextBox>
            <asp:RequiredFieldValidator ID = "rfvLoginName" runat = "server" ErrorMessage = "请输入用户名" ControlToValidate = "txtLoginName">*</asp:RequiredFieldValidator></td></tr>
    <tr><td>密码</td>
        <td><asp:TextBox ID = "txtLoginPwd" runat = "server" TextMode = "Password"></asp:TextBox>
            <asp:RequiredFieldValidator ID = "rfvLoginPwd" runat = "server" ErrorMessage = "请输入密码" ControlToValidate = "txtLoginPwd">*</asp:RequiredFieldValidator></td></tr>
    <tr><td>确认密码</td>
        <td><asp:TextBox ID = "txtPwdAgain" runat = "server" TextMode = "Password"></asp:TextBox>
            <asp:RequiredFieldValidator ID = "rfvPwdAgain" runat = "server" ErrorMessage = "请输入确认密码" ControlToValidate = "txtPwdAgain">*</asp:RequiredFieldValidator>
            <asp:CompareValidator ID = "cvPwdAgain" runat = "server" ErrorMessage = "两次密码不一致" ControlToValidate = "txtPwdAgain" ControlToCompare = "txtLoginPwd">*</asp:CompareValidator>
        </td></tr>
    <tr><td>真实姓名</td>
        <td><asp:TextBox ID = "txtRealName" runat = "server"></asp:TextBox>
            <asp:RequiredFieldValidator ID = "rfvRealName" runat = "server" ErrorMessage = "请输入真实姓名" ControlToValidate = "txtRealName">*</asp:RequiredFieldValidator></td></tr>
    <tr><td>地址</td>
        <td><asp:TextBox ID = "txtAddress" runat = "server"></asp:TextBox></td></tr>
    <tr><td>电话</td>
        <td><asp:TextBox ID = "txtPhone" runat = "server"></asp:TextBox></td></tr>
```

```
<tr><td>电子邮件</td>
    <td><asp:TextBox ID="txtEmail" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="rfvEmail" runat="server" ErrorMessage=
"请输入电子邮件地址" ControlToValidate="txtEmail">*</asp:RequiredFieldValidator>
        <asp:RegularExpressionValidator ID="revEmail" runat="server" ErrorMessage="电
子邮件地址格式错误" ControlToValidate="txtEmail" ValidationExpression="\w+([-+.']\w+)
*@\w+([-.]\w+)*\.\w+([-.]\w+)*">*</asp:RegularExpressionValidator></td></tr>
<tr><td>验证码</td>
    <td><asp:TextBox ID="txtCode" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="rfvCode" runat="server" ErrorMessage="请
输入验证码" ControlToValidate="txtCode">*</asp:RequiredFieldValidator>
        <cc1:SerialNumber ID="snCode" runat="server">
        </cc1:SerialNumber></td></tr>
<tr><td colspan="2" class="content_center">
        <asp:ValidationSummary ID="vsRegister" runat="server" ShowMessageBox=
"true" ShowSummary="false"/></td></tr>
<tr><td colspan="2" class="content_center">
        <asp:Button ID="btnSubmit" runat="server" Text="提交" OnClick="btnSubmit_
Click" />
    </td></tr>
</table>
```

UserRegister.aspx.cs 实现代码如下：

```
...
using NewsModels;
using NewsBLL;
namespace Web.Member
{
    public partial class UserRegister : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                //首次加载生成新验证码
                snCode.Create();
            }
        }
        protected void btnSubmit_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                if (!CheckCode())
                {
                    Common.Message.RegScript(this, "验证码错误!");
                }
                else
                {
                    User user = new User();
```

```
                    user.LoginName = txtLoginName.Text.Trim();
                    user.LoginPwd = txtLoginPwd.Text.Trim();
                    user.RealName = txtRealName.Text.Trim();
                    user.Address = txtAddress.Text.Trim();
                    user.Phone = txtPhone.Text.Trim();
                    user.Email = txtEmail.Text.Trim();
                    user.Role = "1";
                    if (!UserManager.UserRegister(user))
                    {
                        Common.Message.RegScript(this, "用户名已使用!请重新选择!");
                    }
                    else
                    {
                        Common.Message.RegScript(this, "注册成功!请登录!", "UserLogin.
aspx");
                    }
                }
            }
        }
        private bool CheckCode()
        {
            if (snCode.CheckSN(txtCode.Text.Trim()))    //判断验证码输入是否正确
            {
                return true;
            }
            else
            {
                snCode.Create();    //如果验证码输入不正确,则生成新验证码
                return false;
            }
        }
    }
}
```

用户注册页验证效果如图 5-17 所示。



图 5-17　用户注册页验证效果

5.3.4 小结

(1) 向数据库添加数据的基本步骤为：创建 SqlConnection 对象→定义 SQL 语句(INSERT)→创建 SqlCommand 对象→执行 ExecuteNonQuery()方法→根据返回的结果进行处理。

(2) 使用参数化 SQL 语句可使程序结构更加清晰，提高程序的可读性。

5.3.5 思考与练习

1. 用 SqlCommand 对象的 ExecuteNonQuery()方法向数据库添加数据的基本步骤是什么?

2. 如何给参数化 SQL 语句赋参数值?

任务 5.4 使用 GridView 控件实现新闻管理

任务目标

(1) 了解各种数据源控件与数据绑定控件的类型和作用。

(2) 会用 ObjectDataSource 控件从数据源获取数据。

(3) 会用 GridView 控件显示、删除、编辑数据。

5.4.1 数据源控件与数据绑定控件概述

前面介绍了 ASP.NET 数据访问技术中的 ADO.NET 对象，接下来介绍数据绑定技术，使用该技术可以使 Web 应用程序轻松地与数据库进行交互。简单地说，数据绑定是将数据源中的数据取出来，显示在页面的各种控件上，用户可以通过这些控件查看和修改数据。ASP.NET 2.0 拥有一系列实现数据绑定的工具，包括几个数据源控件和数据绑定控件。

1. 数据源控件

ASP.NET 2.0 在 ADO.NET 的数据模型基础上进行了进一步的封装和抽象，提出了一个新的概念——“数据源控件”(DataSource Control)。数据源控件用于实现从不同数据源获取数据的功能，它可以设置连接信息、查询信息、参数和行为，可以消除 ASP.NET 1.x 中要求的大量的重复性代码。数据源控件被放置在 Visual Studio 2005 工具箱的“数据”选项卡中，分别以×××DataSource 命名(如 SqlDataSource、ObjectDataSource 等)。数据源控件的类型及作用如表 5-10 所示。

特别需要指出的是，大多数 ASP.NET 数据源控件，如 SqlDataSource，都在两层应用程序层次结构中使用。在该层次结构中，表示层(ASP.NET 网页)可以与数据层(数据库和 XML 文件等)直接进行通信。但是，常用的应用程序设计原则是，将表示层与业务逻辑相分离，而将业务逻辑封装在业务对象中。这些业务对象在表示层和数据层之间形成一层，从而生成一种三层应用程序结构。ObjectDataSource 控件通过提供一种将相关页上的数据控件绑定到中间层业务对象的方法，为三层结构提供支持。

表 5-10　数据源控件

控 件 名 称	说　　明
SqlDataSource	用于从 SQL Server、OLE DB、ODBC、Oracle 等数据源中检索数据
AccessDataSource	继承自 SqlDataSource，专门用于从 Access 数据库检索数据
ObjectDataSource	可用于三层体系结构，能够将来自业务逻辑层的数据对象与表示层中的数据绑定控件绑定，实现数据的检索或更新
XmlDataSource	用于检索和处理 XML 文件。通过 XmlDataSource 控件可将一个 XML 文件绑定到一个用于显示层次结构的 TreeView 或 Menu 控件上
SiteMapDataSource	结合 ASP.NET 站点导航使用。通过 SiteMapDataSource 控件可将一个站点地图文件.sitemap 绑定到 TreeView 或 Menu 控件上，提供站点导航

2. 数据绑定控件

数据源控件只负责管理与实际数据存储源的连接，并不能呈现任何用户界面，要将数据显示出来，需要数据绑定控件。数据绑定控件是将数据作为标记向发出请求的客户端设备或浏览器呈现的 UI 控件。数据绑定控件包括 GridView、DetailsView、FormView、Repeater、DataList 等。这类控件主要提供数据显示、编辑、删除等相关用户界面。

ASP.NET 2.0 中有以下两种数据绑定方式。

(1) 使用数据源控件

首先使用数据源控件连接数据库，并返回数据集合，然后利用数据绑定控件实现数据显示、更新、删除等功能。

其中，需通过一个重要属性 DataSourceID(所有数据绑定控件共有的属性)将数据源控件和数据绑定控件“连接”起来。可以直接设置该属性，也可以通过编写代码实现。语法格式为：

```
数据绑定控件 ID.DataSourceID = 数据源控件 ID;
```

(2) 编码指定数据源

即编写代码在程序运行中动态绑定数据源。比如：

```
gvUsers.DataSource = NewsBLL.UserManager.GetUsers();
gvUsers.DataBind();
```

其中，gvUsers 为数据绑定控件 ID，GetUsers()返回一个数据集合作为数据源。

5.4.2　GridView 控件简介

GridView 控件以表的形式显示数据，每一列代表一个字段，每一行代表一条记录。可配合数据源控件对数据库进行浏览、编辑、删除等操作。GridView 控件中的数据显示格式既可以套用已存在的格式，也可以通过属性来设置，包括 GridView 控件行的布局、颜色、字体、对齐方式以及指定行中包含的文本和数据的显示。GridView 控件的常用属性与事件如表 5-11 所示。

表 5-11　GridView 控件的常用属性与事件

属　　性	说　　明
AutoGenerateColumns	获取或设置一个值,该值指示是否为数据源中的每个字段自动创建绑定字段
AllowPaging	获取或设置一个值,该值指示是否启用分页功能
PageSize	获取或设置 GridView 控件在每页上所显示的记录数目
DataKeyNames	获取或设置一个数组,该数组包含了显示在 GridView 控件中的项的主键字段的名称
DataSource	获取或设置对象,数据绑定控件从该对象中检索其数据项列表
DataSourceID	获取或设置控件的 ID,数据绑定控件从该控件中检索其数据项列表
事　　件	**说　　明**
DataBound	在 GridView 控件完成到数据源的绑定后发生
RowDataBound	在 GridView 控件中的某个行被绑定到一个数据记录时发生
PageIndexChanging	单击“页导航”按钮时,在 GridView 控件执行分页操作之前发生
RowDeleting	单击 GridView 控件内某一行的“删除”按钮时,在 GridView 控件从数据源删除该行记录之前发生
RowDeleted	单击 GridView 控件内某一行的“删除”按钮时,在 GridView 控件从数据源删除该行记录之后发生
RowEditing	单击 GridView 控件内某一行的“编辑”按钮时,在 GridView 控件进入编辑模式之前发生
RowCancelingEdit	单击 GridView 控件内某一行的“编辑”按钮时,在 GridView 控件退出编辑模式之前发生
RowUpdating	单击 GridView 控件内某一行的“更新”按钮时,在 GridView 控件更新记录之前发生
RowUpdated	单击 GridView 控件内某一行的“更新”按钮时,在 GridView 控件更新记录之后发生

5.4.3　新闻类别管理数据访问层与业务逻辑层的实现

要实现新闻类别的显示、编辑、删除功能,仍需从数据访问层、业务逻辑层和表示层分别进行编码。

1. 新闻类别管理数据访问层的实现

在 NewsDAL 项目中新建类文件 NewsCategoryService.cs,其关键代码如下:

```
…
using NewsModels;
namespace NewsDAL
{
    public static class NewsCategoryService
    {
        private static string connectionString = ConfigurationManager.ConnectionStrings
["NewsDBConnectionString"].ConnectionString;
        /// <summary>
        /// 删除新闻类别
        /// </summary>
        /// <param name="newsCategory">新闻类别对象</param>
```

```
public static void DeleteNewsCategory(NewsCategory newsCategory)
{
    using (SqlConnection cn = new SqlConnection(connectionString))
    {
        cn.Open();
        SqlCommand cm = new SqlCommand();
        cm.Connection = cn;
        string sql = "DELETE FROM NewsCategories WHERE Id = @Id";
        cm.CommandText = sql;
        cm.Parameters.AddWithValue("@Id", newsCategory.Id);
        cm.ExecuteNonQuery();
    }
}
/// <summary>
/// 修改新闻类别
/// </summary>
/// <param name="newsCategory">新闻类别对象</param>
public static void ModifyNewsCategory(NewsCategory newsCategory)
{
    using (SqlConnection cn = new SqlConnection(connectionString))
    {
        cn.Open();
        SqlCommand cm = new SqlCommand();
        cm.Connection = cn;
        string sql =
        "UPDATE NewsCategories " +
        "SET " +
            "Name = @Name " +
        "WHERE Id = @Id";
        cm.CommandText = sql;
        cm.Parameters.AddWithValue("@Id", newsCategory.Id);
        cm.Parameters.AddWithValue("@Name", newsCategory.Name);
        cm.ExecuteNonQuery();
    }
}
/// <summary>
/// 查询所有新闻类别的所有字段信息
/// </summary>
/// <returns>新闻类别对象集合</returns>
public static IList<NewsCategory> GetNewsCategories()
{
    using (SqlConnection cn = new SqlConnection(connectionString))
    {
        cn.Open();
        SqlCommand cm = new SqlCommand();
        cm.Connection = cn;
        string sql = "SELECT * FROM NewsCategories";
        cm.CommandText = sql;
        SqlDataReader dr = cm.ExecuteReader();
```

```
                //使用 List<T>传递实体对象集合
                List<NewsCategory> list = new List<NewsCategory>();
                while (dr.Read())
                {
                    NewsCategory newsCategory = new NewsCategory();
                    newsCategory.Id = (int)dr["Id"];
                    newsCategory.Name = (string)dr["Name"];
                    list.Add(newsCategory);
                }
                dr.Close();
                return list;
            }
        }
    }
}
```

2. 新闻类别管理业务逻辑层的实现

在 NewsBLL 项目中新建类文件 NewsCategoryManager.cs,其关键代码如下:

```
…
using NewsModels;
using NewsDAL;
namespace NewsBLL
{
    public static class NewsCategoryManager
    {
        public static void DeleteNewsCategory(NewsCategory newsCategory)
        {
            NewsCategoryService.DeleteNewsCategory(newsCategory);
        }
        public static void ModifyNewsCategory(NewsCategory newsCategory)
        {
            NewsCategoryService.ModifyNewsCategory(newsCategory);
        }
        public static IList<NewsCategory> GetNewsCategories()
        {
            return NewsCategoryService.GetNewsCategories();
        }
    }
}
```

5.4.4 使用 GridView 控件实现新闻类别显示

在新闻系统中,管理员后台需具备用户管理、新闻类别管理、新闻文章管理、留言管理等管理功能,但对页面的美观要求不是很高。所以,后台管理页面通常可使用 GridView 控件进行列表显示。

在 Web 项目的 Admin 文件夹下,新建页面文件 NewsCategoriesManager.aspx。

使用 GridView 控件显示新闻类别的基本步骤如下。

（1）添加数据源控件。切换到“设计”视图，将工具箱“数据”选项卡中的 ObjectDataSource 控件拖放到页面中，设置 ObjectDataSource 控件的 ID 属性值为 odsNewsCategories。单击 ObjectDataSource 右上角的小三角按钮，在弹出的列表中选择“配置数据源”选项，如图 5-18 所示。在弹出的“配置数据源-odsNewsCategories”对话框中，选择“选择业务对象”下拉列表的 NewsBLL. NewsCategoryManager 选项，如图 5-19 所示。单击“下一步”按钮，打开图 5-20 所示的对话框定义各种数据方法。在 SELECT 选项卡的“选择方法”下拉列表中选择“GetNewsCategories()，返回 IList<NewsCategory>”选项。



图 5-18　添加到页面设计视图中的数据源控件



图 5-19　选择业务对象



图 5-20　定义数据方法——配置 SELECT 操作

(2) 添加 GridView 控件。将工具箱“数据”选项卡中的 GridView 控件拖放到页面中，设置 GridView 控件的 ID 属性值为 gvNewsCategories。单击 GridView 右上角的小三角按钮，在“选择数据源”下拉列表中选择前面创建的数据源 odsNewsCategories，将数据源绑定到 GridView 控件，如图 5-21 所示。或者将 gvNewsCategories 的 DataSourceID 属性值设置为 odsNewsCategories 完成相同功能。在选择了数据源后，GridView 任务菜单中将多出若干选项，如图 5-22 所示。若希望程序具有“分页”、“删除”、“编辑”等数据库操作功能可选择相应的复选框。此时，已经完成了 GridView 控件最简单的数据绑定。运行页面，GridView 控件显示新闻类别列表初步效果如图 5-23 所示。可以对 GridView 控件的显示效果再做改进，例如使 GridView 的界面更加美观，把列表标题改成中文显示并隐藏 Id 列、当记录数太多时能分页显示等。下面在此基础上使用 GridView 控件其他的一些功能。

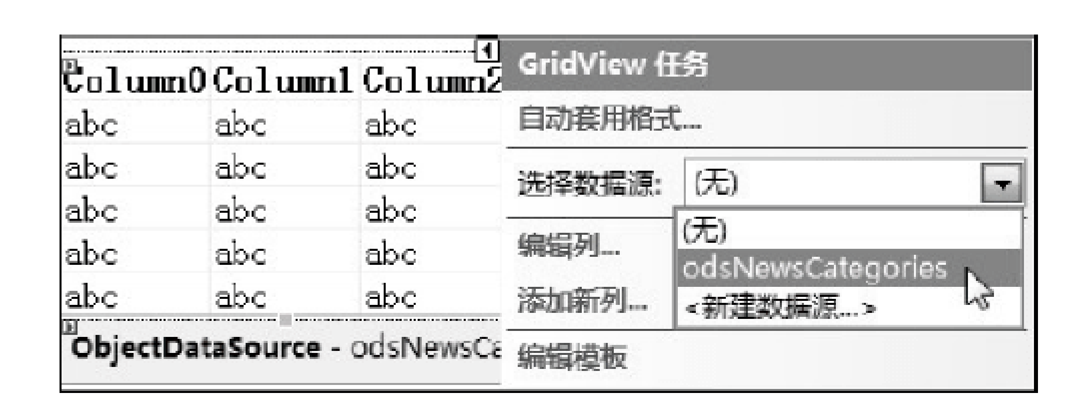


图 5-21 将数据源绑定到 GridView 控件



图 5-22 GridView 任务菜单



Id	Name
1	财经
2	体育
3	娱乐
4	教育
5	科技
6	军事

图 5-23 用 GridView 显示新闻类别初步效果的运行界面

(3) 设置 GridView 控件的外观。单击 GridView 右上角的小三角按钮，在如图 5-22 所示的 GridView 任务菜单中，选择“自动套用格式”命令打开如图 5-24 所示的对话框，可在“选择方案”列表中任选一种显示格式(如“专业型”)改变 GridView 外观。

(4) 自定义 GridView 控件显示列。在如图 5-22 所示的 GridView 任务菜单中，选择“编辑列”命令打开“字段”对话框，为了能手动选择需要绑定的列，先去掉左下方“自动生成字段”复选框前面的“√”。然后，在左上方“可用字段”列表中选择 BoundField 选项，单击“添加”按钮将其添加到“选定的字段”列表，在右边“BoundField 属性”列表中，设置该



图 5-24　“自动套用格式”对话框

绑定列的 DataField 属性为 Id、HeaderText 属性为 Id、Visible 属性为 False。类似地，再添加另一个绑定列，设置该绑定列的 DataField 属性为 Name、HeaderText 属性为“类别名称”。绑定列设置结果如图 5-25 所示，有关绑定列的常用属性如表 5-12 所示。此时，已经实现了所有新闻类别的显示。从如图 5-26 所示页面运行效果来看，虽然 GridView 上绑定了字段 Id，却没有显示在页面上。通过手动设置绑定列，可以达到自由绑定或显示列的目的。

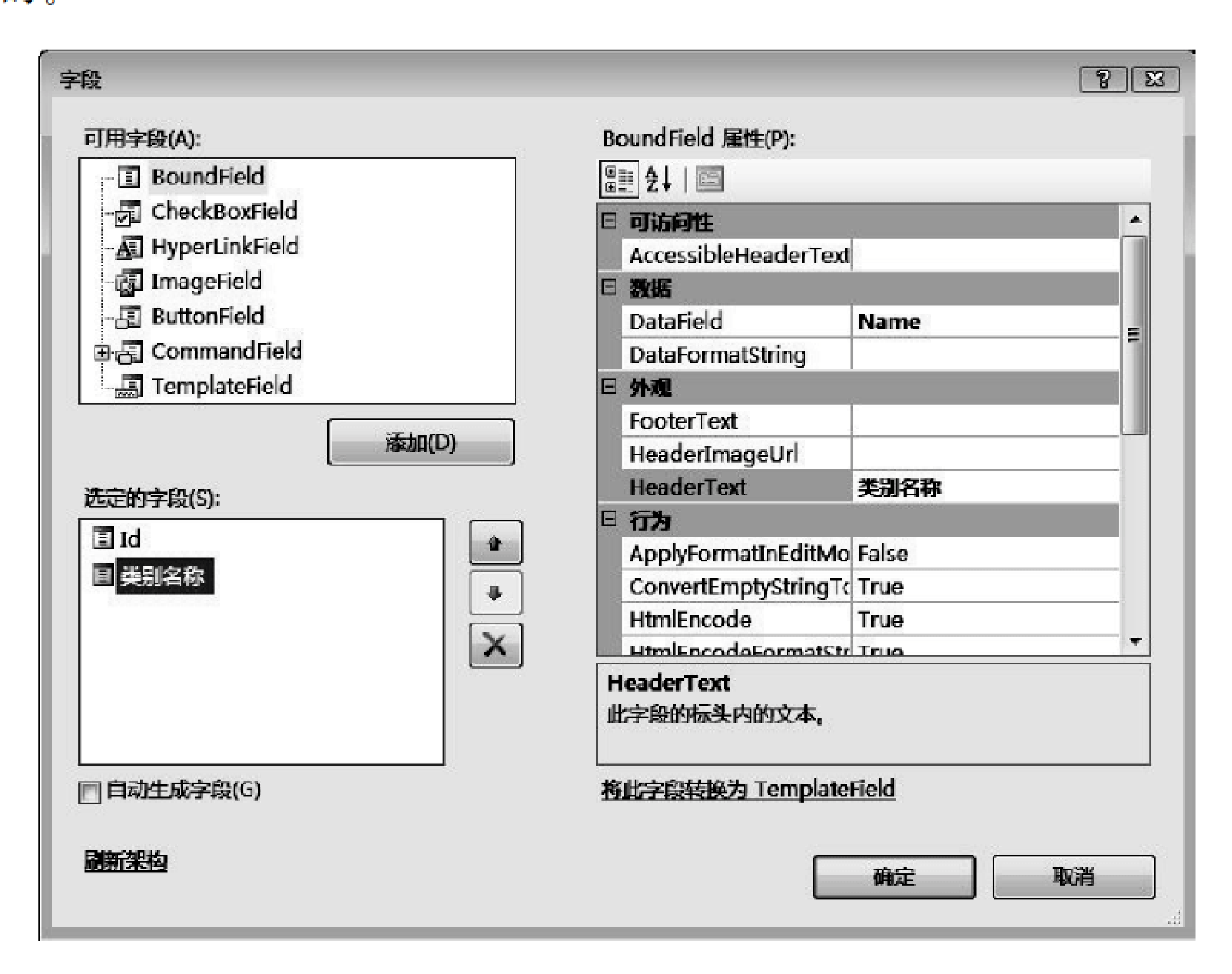


图 5-25　绑定字段

(5) 启用分页。在图 5-22 所示的 GridView 任务菜单中,选中“启用分页”复选框。或者将 GridView 的 AllowPaging 属性改成 True 完成相同功能。在默认情况下,每页显示 10 条记录,可通过 PageSize 属性修改。在“设计”视图下,设置分页后的 GridView 如图 5-27 所示。

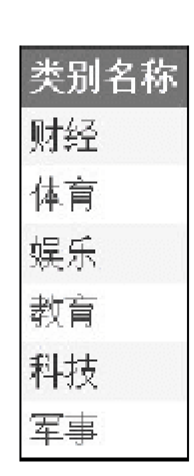


图 5-26　用 GridView 显示新闻类别列表运行界面



图 5-27　设置分页后在“设计”视图下的 GridView

表 5-12　BoundField 列的常用属性

属　　性	说　　明
DataField	获取或设置要绑定到 BoundField 对象的数据字段的名称
HeaderText	获取或设置显示在数据控件标头中的文本
Visible	获取或设置指示是否呈现数据控件字段的值
DataFormatString	获取或设置字符串,该字符串指定字段值的显示格式。例如,格式化字符串“{0:F2}”将显示带两位小数的定点数。注意,使用 DataFormatString 属性时,必须将 HtmlEncode 属性设置为 False,否则设置无效
HtmlEncode	获取或设置一个值,该值指示在 BoundField 对象中显示字段值之前,是否对这些字段值进行 HTML 编码
ReadOnly	获取或设置一个值,该值指示是否可以在编辑模式中修改字段的值

5.4.5　使用 GridView 控件实现新闻类别的编辑、删除

在默认情况下,GridView 控件在只读模式下显示数据。但是,该控件还支持一种编辑模式,在该模式下控件显示一个包含可编辑控件(如 TextBox)的行。除此之外,还可以对 GridView 控件进行配置以显示一个删除按钮,用户可单击该按钮来删除数据源中相应的记录。下面继续给 NewsCategoriesManager.aspx 页面增加编辑和删除功能。

使用 GridView 控件编辑、删除新闻类别的基本步骤如下。

(1) 在数据源控件定义数据方法。切换到“设计”视图,打开如图 5-20 所示的对话框,打开 UPDATE 选项卡选择 ModifyNewsCategory(NewsCategory newsCategory)方法,打开 DELETE 选项卡选择 DeleteNewsCategory(NewsCategory newsCategory)方法。

(2) 设置 DataKeyNames 属性。设置 GridView 控件的 DataKeyNames 属性为 Id。

注意: 使用 DataKeyNames 属性指定表示数据源主键的字段。必须设置该属性,否则 GridView 控件的自动更新和删除功能将不起作用。这也是前面做 GridView 显示功

能时，虽不想在用户界面上出现 Id，却要在页面上绑定 Id 的原因。

(3) 添加编辑列与删除列。在如图 5-22 所示的 GridView 任务菜单中，选择“编辑列”命令打开“字段”对话框，在左上方“可用字段”列表中将 CommandField 前的“+”号展开，分别选择“编辑、更新、取消”选项与“删除”选项添加到“选定的字段”列表中，如图 5-28 所示。程序运行后显示如图 5-29 所示的界面，如果单击页面中“删除”按钮，则所在行的数据记录将直接从数据库中删除。如果单击页面中“编辑”按钮，则页面切换为如图 5-30 所示的编辑模式，在修改了数据后可单击“更新”按钮将现有数据保存到数据库中，单击“取消”按钮则放弃对数据的修改。

(4) 设置模板列。数据绑定控件的模板列可用于显示用户自定义内容。比如想在编辑时进行类别名称输入的非空验证，或是在删除前加上一个确认删除的对话框，可结合将有关列转换为模板列来实现。打开如图 5-28 所示的“字段”对话框，在左下方“选定的字段”列表中，选择“类别名称”选项，单击右下方“将此字段转换为 TemplateField”超链接将它转换为模板列。在如图 5-22 所示的 GridView 任务菜单中，选择“编辑模板”进入 GridView 的模板编辑模式，在“显示”下拉列表中选择“Column[1]-类别名称”以展开显示所有的模板项，向 EditItemTemplate 模板项设置一个数据验证控件 RequiredFieldValidator，使得能够对该模板项中的文本框控件进行非空验证，如图 5-31 所示。类似地，将“字段”对话框中的 CommandField“删除”列也转换为 TemplateField 列，并给“删除”按钮添加一个客户端事件 OnClientClick，写上确认处理的 JavaScript 脚本：OnClientClick＝"return confirm('确认要删除吗?');"。这样，执行删除操作时会在网页上弹出一个对话框，先让用户确认，然后再删除记录，以避免误操作引起的误删除。有关 TemplateField 列的各个模板项作用如表 5-13 所示。

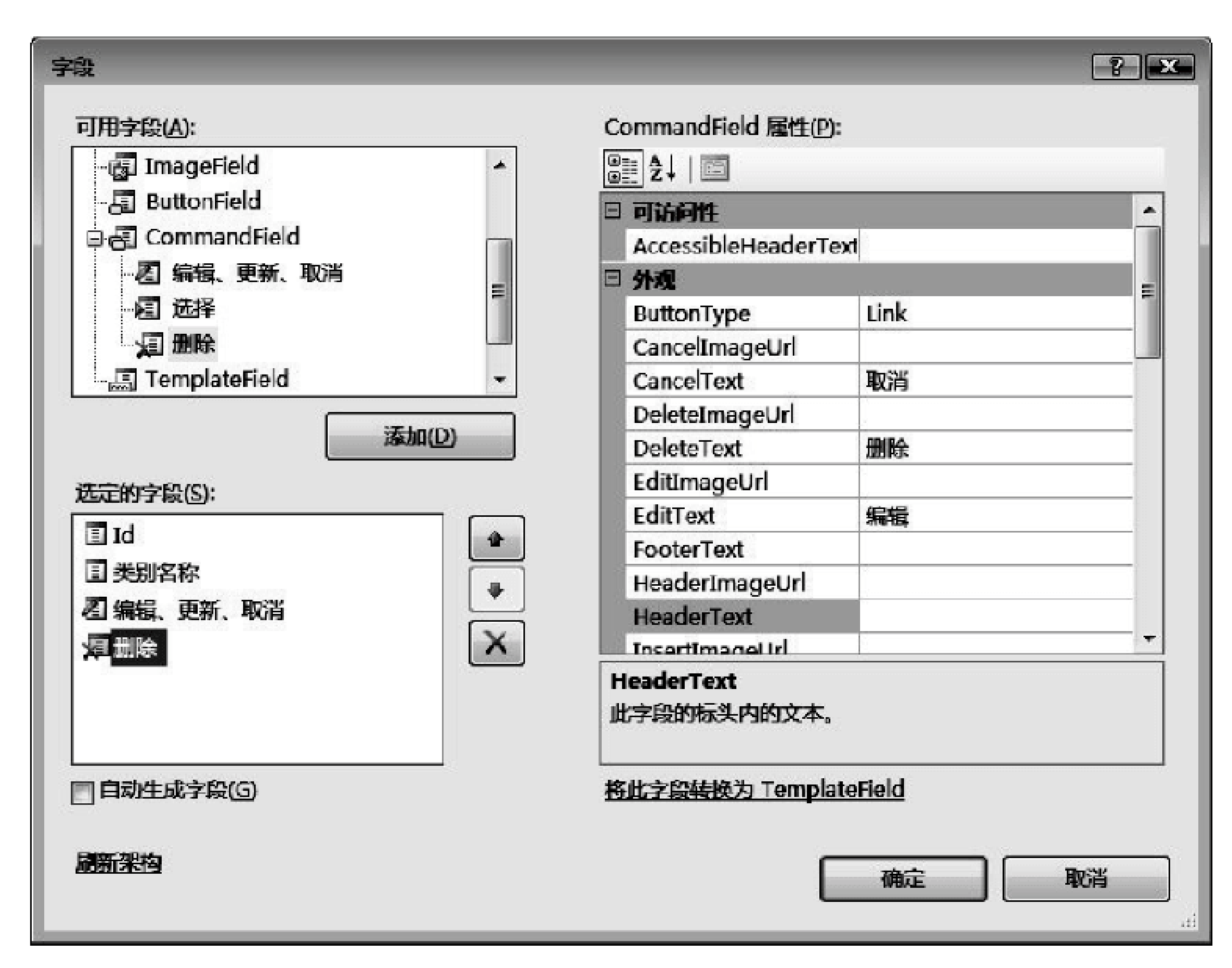


图 5-28　添加编辑列与删除列

类别名称		
财经	编辑	删除
体育	编辑	删除
娱乐	编辑	删除
教育	编辑	删除
科技	编辑	删除
军事	编辑	删除

图 5-29　GridView 添加编辑列与删除列后的运行界面

类别名称		
财经	更新 取消	
体育	编辑	删除
娱乐	编辑	删除
教育	编辑	删除
科技	编辑	删除
军事	编辑	删除

图 5-30　GridView 编辑状态运行界面



图 5-31　添加编辑列

表 5-13　TemplateField 列的各模板项说明

模 板 属 性	说　　明
ItemTemplate	为 TemplateField 对象中的项指定要显示的内容
AlternatingItemTemplate	为 TemplateField 对象中的交替项指定要显示的内容
EditItemTemplate	为 TemplateField 对象中处于编辑模式中的项指定要显示的内容
InsertItemTemplate	为 TemplateField 对象中处于插入模式中的项指定要显示的内容。只有 DetailsView 控件支持该模板
HeaderTemplate	为 TemplateField 对象的标头部分指定要显示的内容
FooterTemplate	为 TemplateField 对象的脚注部分指定要显示的内容

新闻类别管理页 NewsCategoriesManager.aspx 的部分 HTML 代码如下所示。

```
<%@ Page Language = "C#" MasterPageFile = "~/Admin/Admin.Master" AutoEventWireup = "true"
CodeBehind = "NewsCategoriesManager.aspx.cs" Inherits = "Web.Admin.NewsCategoriesManager" Title =
"类别管理" %>
<asp:Content ID = "Content1" ContentPlaceHolderID = "ContentPlaceHolder1" runat = "server">
```

```
    <div id = "admin_div">
            <asp: GridView ID = "gvNewsCategories" runat = "server" DataSourceID =
"odsNewsCategories" AutoGenerateColumns = "False" CellPadding = "4" ForeColor = "#333333"
GridLines = "None" AllowPaging = "True" DataKeyNames = "Id">
            <Columns>
                <asp:BoundField DataField = "Id" HeaderText = "Id" Visible = "False" />
                <asp:TemplateField HeaderText = "类别名称">
                    <EditItemTemplate>
                        <asp:TextBox ID = "txtName1" runat = "server" Text = '<%# Bind
("Name") %>'></asp:TextBox><asp:RequiredFieldValidator ID = "rfvName" runat = "server"
ErrorMessage = "类别名称不能为空" ControlToValidate = "txtName1"></asp:RequiredFieldValidator>
                    </EditItemTemplate>
                    <ItemTemplate>
                        <asp:Label ID = "Label1" runat = "server" Text = '<%# Bind("Name")
%>'></asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:TemplateField ShowHeader = "False">
                    <ItemTemplate>
                        <asp:LinkButton ID = "LinkButton1" runat = "server" CausesValidation =
"False" CommandName = "Delete"
                            Text = "删除" OnClientClick = "return confirm('确认要删除
吗?');"></asp:LinkButton>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:CommandField ShowEditButton = "True" />
            </Columns>
            <RowStyle BackColor = "#F7F6F3" ForeColor = "#333333" />
            <FooterStyle BackColor = "#5D7B9D" Font-Bold = "True" ForeColor = "White" />
            <PagerStyle BackColor = "#284775" ForeColor = "White" HorizontalAlign = "Center" />
            <SelectedRowStyle BackColor = "#E2DED6" Font-Bold = "True" ForeColor = "#333333" />
            <HeaderStyle BackColor = "#5D7B9D" Font-Bold = "True" ForeColor = "White" />
            <EditRowStyle BackColor = "#999999" />
            <AlternatingRowStyle BackColor = "White" ForeColor = "#284775" />
        </asp:GridView>
        <asp: ObjectDataSource ID = "odsNewsCategories" runat = "server" SelectMethod =
"GetNewsCategories" TypeName = "NewsBLL.NewsCategoryManager" DataObjectTypeName = "NewsModels.
NewsCategory" DeleteMethod = "DeleteNewsCategory" UpdateMethod = "ModifyNewsCategory"> </asp:
ObjectDataSource>
    </div>
</asp:Content>
```

新闻类别管理页最终运行效果如图 5-32 所示。

5.4.6　新闻列表显示、删除数据访问层与业务逻辑层的实现

新闻列表显示数据访问层与业务逻辑层的工作主要就是对新闻表(News)进行查询并返回一个新闻列表对象集合。

注意：新闻列表显示时需要显示新闻类别名称，而新闻类别(NewsCategoryId)字段



图 5-32　新闻类别管理页最终运行效果

在新闻表(News)中是一个以整数形式存在的外键，所以要显示新闻类别名称，还得查询新闻类别表(NewsCategories)。另外，由于新闻表的新闻内容(Contents)字段值的字数太多，通常在新闻列表中不用显示该字段，所以查询时只取出部分字段。还有，为了便于新闻管理，查询时做了搜索和排序功能。下面针对新闻的列表显示和删除功能给出数据访问层与业务逻辑层中的实现方法。

1. 新闻类别管理数据访问层、业务逻辑层增加新方法

在 NewsDAL 项目的 NewsCategoryService 类中，增加一个根据新闻类别 Id 查询并返回新闻类别对象的方法。实现代码如下：

```
/// <summary>
/// 根据 Id 查询新闻类别
/// </summary>
/// <param name="id">新闻类别 Id</param>
/// <returns>新闻类别对象</returns>
public static NewsCategory GetNewsCategoryById(int id)
{
    using (SqlConnection cn = new SqlConnection(connectionString))
    {
        cn.Open();
        SqlCommand cm = new SqlCommand();
        cm.Connection = cn;
        string sql = "SELECT * FROM NewsCategories WHERE Id = @Id";
        cm.CommandText = sql;
        cm.Parameters.AddWithValue("@Id", id);
        SqlDataReader dr = cm.ExecuteReader();
        if (dr.Read())
        {
            NewsCategory newsCategory = new NewsCategory();
            newsCategory.Id = (int)dr["Id"];
```

```
                newsCategory.Name = (string)dr["Name"];
                dr.Close();
                return newsCategory;
            }
            else
            {
                dr.Close();
                return null;
            }
        }
    }
```

在 NewsBLL 项目的 NewsCategoryManager 类中，增加相应方法。实现代码如下：

```
public static NewsCategory GetNewsCategoryById(int id)
{
    return NewsCategoryService.GetNewsCategoryById(id);
}
```

2. 新闻列表显示、删除数据访问层的实现

在 NewsDAL 项目中新建类文件 NewsService.cs 作为新闻文章管理的数据访问层，在其中编写与新闻列表显示和删除功能有关的方法。实现代码如下：

```
...
using NewsModels;
namespace NewsDAL
{
    public static class NewsService
    {
        private static string connectionString = ConfigurationManager.ConnectionStrings
["NewsDBConnectionString"].ConnectionString;
            /// <summary>
            /// 删除新闻
            /// </summary>
            /// <param name="news">新闻对象</param>
            public static void DeleteNews(News news)
            {
                using (SqlConnection cn = new SqlConnection(connectionString))
                {
                    cn.Open();
                    SqlCommand cm = new SqlCommand();
                    cm.Connection = cn;
                    string sql = "DELETE FROM News WHERE Id = @Id";
                    cm.CommandText = sql;
                    cm.Parameters.AddWithValue("@Id", news.Id);
                    cm.ExecuteNonQuery();
                }
            }
            /// <summary>
            /// 根据 SQL 语句返回部分字段的新闻列表
```

```
        /// </summary>
        /// <param name="sql">SQL 语句</param>
        /// <returns>新闻对象集合</returns>
        private static IList<News> GetNewsBySql(string sql)
        {
            using (SqlConnection cn = new SqlConnection(connectionString))
            {
                cn.Open();
                SqlCommand cm = new SqlCommand();
                cm.Connection = cn;
                cm.CommandText = sql;
                SqlDataReader dr = cm.ExecuteReader();
                List<News> list = new List<News>();
                while (dr.Read())
                {
                    News news = new News();
                    for (int i = 0; i < dr.FieldCount; i++)
                    {
                        string fieldName = dr.GetName(i);
                        if (fieldName == "Id")
                            news.Id = (int)dr["Id"];
                        else if (fieldName == "Title")
                            news.Title = (string)dr["Title"];
                        else if (fieldName == "Author")
                            news.Author = (string)dr["Author"];
                        else if (fieldName == "PubDate")
                            news.PubDate = (DateTime)dr["PubDate"];
                        else if (fieldName == "Contents")
                            news.Contents = (string)dr["Contents"];
                        else if (fieldName == "Clicks")
                            news.Clicks = (int)dr["Clicks"];
                        else if (fieldName == "NewsCategoryId")
                        {
                            news.NewsCategoryId = (int)dr["NewsCategoryId"];
                            news. NewsCategory = NewsCategoryService. GetNewsCategoryById
((int)dr["NewsCategoryId"]);
                        }
                    }
                    list.Add(news);
                }
                dr.Close();
                return list;
            }
        }
        /// <summary>
        /// 根据查询条件、排序字段、排序方向返回包含部分字段(Id,Title,Author,PubDate,
Clicks,NewsCategoryId)的新闻列表
        /// </summary>
        /// <param name="conditions">查询条件</param>
```

```
        /// <param name = "sortField">排序字段</param>
        /// <param name = "direction">排序方向</param>
        /// <returns>新闻对象集合</returns>
        public static IList < News > GetNewsPartFieldsByConditions ( string conditions,
string sortField, string direction)
        {
            string sql = "SELECT Id, Title, Author, PubDate, Clicks, NewsCategoryId FROM News";
            if (conditions.Trim().Length > 0)
            {
                sql += " WHERE " + conditions;
            }
            if (sortField.Trim().Length > 0)
            {
                sql += " ORDER BY " + sortField;
            }
            if (direction.Trim().Length > 0)
            {
                sql += " " + direction;
            }
            return GetNewsBySql(sql);
        }
    }
}
```

3. 新闻列表显示、删除业务逻辑层的实现

在 NewsBLL 项目中新建类文件 NewsManager.cs 作为新闻文章管理的业务逻辑层，在其中编写新闻列表显示和删除的相应方法。实现代码如下：

```
…
using NewsModels;
using NewsDAL;
namespace NewsBLL
{
    public static class NewsManager
    {
        public static void DeleteNews(News news)
        {
            NewsService.DeleteNews(news);
        }
        public static IList < News > GetNewsPartFieldsByConditions ( string conditions,
string sortField, string direction)
        {
            return NewsService.GetNewsPartFieldsByConditions(conditions, sortField, direction);
        }
    }
}
```

5.4.7　使用 GridView 控件实现新闻列表的显示、删除

管理员后台的新闻文章管理页面需要以列表方式显示新闻，同样也可使用 GridView

控件实现。与前面新闻类别管理页面相比，删除操作的处理方法是一样的，在此不再赘述。不同之处在于，GridView 表格里显示新闻类别名称时，要注意对外键对象 NewsCategory 的处理方法。另外，GridView 表格里只列出部分字段，新闻内容字段通常放在新闻详细页显示，同时可在新闻详细页做新闻编辑功能（有关新闻详细页设计参见任务 5.6）。还有，为了便于新闻管理，在新闻文章管理页做了搜索和排序功能（参见任务 5.5）。下面先为新闻文章管理表示层完成新闻列表的显示功能。

在 Web 项目的 Admin 文件夹下，新建页面文件 NewsManager.aspx。

使用 GridView 控件显示新闻列表的基本步骤如下。

（1）设置 3 个隐藏域控件。切换到“设计”视图，从工具箱“标准”选项卡向页面拖放 3 个 HiddenField 控件，设置其 ID 属性值分别为 hfSearch、hfSortField、hfDirection，用于保存查询条件、排序字段和排序方向的值。并给这些控件赋初值，其 Value 属性值分别设为“1=1”、“Id”、“DESC”。

（2）添加数据源控件。向页面拖放一个 ObjectDataSource 控件，设置其 ID 属性值为 odsNews。配置数据源时，指定选择数据的方法是业务逻辑层 NewsManager 类的 GetNewsPartFieldsByConditions()方法，用于获取新闻数据。其查询参数值来自 3 个隐藏域控件，设置方式如图 5-33 所示。



图 5-33　为新闻列表显示的 SELECT 方法定义参数

（3）设置 GridView 控件各字段。在数据源各字段中，由于新闻类别字段是一个对象，绑定时用模板列实现，绑定值设为 Text='<%# Bind("NewsCategory.Name") %>'。另外，给 GridView 设置一个超链接列 HyperLinkField，单击时可跳转到新闻详细页显示当前记录的详细信息。HyperLinkField 列的常用属性如表 5-14 所示。

表 5-14　HyperLinkField 列的常用属性

属　　性	说　　明
DataNavigateUrlFields	获取或设置数据源中字段的名称，用于为 HyperLinkField 对象中的超链接构造 URL
DataNavigateUrlFormatString	获取或设置用于指定格式的字符串，HyperLinkField 对象中的超链接的 URL 将以此格式呈现
DataTextField	获取或设置数据源中的字段的名称，此名称包含要为 HyperLinkField 对象中的超链接标题显示的文本
DataTextFormatString	获取或设置用于指定格式的字符串，HyperLinkField 对象中的超链接标题将以此格式显示
Text	获取或设置要为 HyperLinkField 对象中的每个超链接显示的文本

新闻文章管理页 NewsManager.aspx 的部分 HTML 代码如下所示：

```
<asp:HiddenField ID = "hfSortField" runat = "server" Value = "Id" />
<asp:HiddenField ID = "hfDirection" runat = "server" Value = "DESC" />
<asp:HiddenField ID = "hfSearch" runat = "server" Value = "1 = 1" />
<asp:GridView ID = "gvNews" runat = "server" AutoGenerateColumns = "False" DataSourceID =
"odsNews" DataKeyNames = "Id" AllowPaging = "True">
    <Columns>
        <asp:BoundField DataField = "Id" HeaderText = "Id" Visible = "False" />
        <asp:BoundField DataField = "Title" HeaderText = "标题" />
        <asp:BoundField DataField = "Author" HeaderText = "作者" />
        <asp:BoundField DataField = "PubDate" HeaderText = "日期" />
        <asp:BoundField DataField = "Clicks" HeaderText = "浏览次数" />
        <asp:TemplateField HeaderText = "类别">
            <ItemTemplate>
                <asp:Label ID = "Label1" runat = "server" Text = '<% # Bind("NewsCategory.
Name") %>'></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField ShowHeader = "False">
            <ItemTemplate>
                <asp:LinkButton ID = "LinkButton1" runat = "server" CausesValidation = "False"
CommandName = "Delete" Text = "删除" OnClientClick = "return confirm('确认要删除吗?');"></asp:
LinkButton>
            </ItemTemplate>
        </asp:TemplateField>
        <asp:HyperLinkField Text = "详细" DataNavigateUrlFields = "Id" DataNavigateUrlFormat-
String = "EditNews.aspx?Id = {0}" />
    </Columns>
</asp:GridView>
<asp:ObjectDataSource ID = "odsNews" runat = "server" DataObjectTypeName = "NewsModels.News"
DeleteMethod = "DeleteNews" SelectMethod = "GetNewsPartFieldsByConditions" TypeName = "NewsBLL.
NewsManager">
    <SelectParameters>
        <asp:ControlParameter ControlID = "hfSearch" Name = "conditions" PropertyName =
```

```
"Value"
                Type = "String" />
            <asp:ControlParameter ControlID = "hfSortField" Name = "sortField" PropertyName =
"Value"
                Type = "String" />
            <asp:ControlParameter ControlID = "hfDirection" Name = "direction" PropertyName =
"Value"
                Type = "String" />
    </SelectParameters>
</asp:ObjectDataSource>
```

用 GridView 控件显示新闻列表运行效果如图 5-34 所示。

标题	作者	日期	浏览次数	类别		
第四届"职场之星"模拟求职大赛	青木	2009/8/19 14:51:20	4	教育	删除	详细
100多家单位聚首温职院"招贤纳士"	乔仁慧	2009/8/19 14:40:52	2	教育	删除	详细
以旧换新拉动新车消费	吴晓	2009/7/2 15:00:00	0	财经	删除	详细
留学要以"兴趣为先"	张凡	2009/7/2 14:49:00	0	教育	删除	详细
公告牌Hot100单曲榜50周年	叶芸	2009/7/2 11:01:00	0	娱乐	删除	详细
《冰河世纪3》香港首映	tungstar	2009/7/2 10:40:00	0	娱乐	删除	详细
中国女队跻身接力决赛	Belldandy	2009/7/2 10:39:00	0	体育	删除	详细
盘点新世纪网坛10冷门	王卓	2009/7/2 10:38:00	0	体育	删除	详细
多数上市银行已超额完成全年信贷目标	钟正	2009/7/2 10:37:00	0	财经	删除	详细
中国（深圳）国际金融博览会今开幕	兴亚	2009/7/2 10:35:00	0	财经	删除	详细
1 2						

图 5-34　用 GridView 控件显示新闻列表运行效果

5.4.8　小结

（1）ASP.NET 2.0 极大地简化了数据库访问，可以通过数据源控件和数据绑定控件展示数据。其中数据源控件提供数据，数据绑定控件提供展示。

（2）当系统为三层结构时，可以将中间层的逻辑功能封装到 ObjectDataSource 控件中。作为数据绑定控件的数据接口，可以在 ObjectDataSource 控件中定义查询、更新、插入、删除等方法，供数据绑定控件调用，使这些控件在 ASP.NET 网页上显示和编辑中间层业务对象中的数据。

（3）在使用实体类开发三层结构，且用户的请求需要返回实体对象集合时，可使用 List＜T＞实现。

（4）GridView 控件按照数据源中的一行显示为输出表中的一行的规则以表的形式显示数据，并提供分页以及编辑或删除单个记录等功能。

（5）数据绑定控件的模板列可用于显示用户自定义内容。它有两种添加方式：直接添加或者将现有字段转换为 TemplateField。

5.4.9　思考与练习

1. 用 ObjectDataSource 控件和 GridView 控件实现管理员后台用户管理列表显示。
2. 用 ObjectDataSource 控件和 GridView 控件实现管理员后台留言管理列表显示。

任务 5.5　使用 DropDownList 控件分类显示新闻

任务目标

（1）会用 DropDownList 控件绑定到数据源显示数据。

（2）会构造复合查询条件进行数据查询。

5.5.1　DropDownList 控件简介

DropDownList 控件是常用的数据绑定控件之一，它首先预定一些列表项，允许用户从预定义的列表中选择一项，除默认选项外，其他列表项在用户单击下拉列表之前一直保持隐藏状态，而且不支持多选功能。DropDownList 控件的常用属性与事件如表 5-15 所示。

表 5-15　DropDownList 控件的常用属性与事件

属　性	说　明
AutoPostBack	获取或设置一个值，该值指示当用户更改列表中的选定内容时是否自动产生向服务器的回发。默认情况下是 false
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DataSourceID	获取或设置控件的 ID，数据绑定控件从该控件中检索其数据项列表
DataTextField	获取或设置为列表项提供文本内容的数据源字段
DataValueField	获取或设置为列表项提供值的数据源字段
Items	获取列表控件项的集合
SelectedIndex	获取或设置 DropDownList 控件中的选定项的索引
SelectedItem	获取列表控件中索引最小的选定项。它常用的两个属性是 Text 和 Value。Text 用于获取或设置项的显示文本，Value 用于获取或设置项的值
SelectedValue	获取列表控件中选定项的值，或选择列表控件中包含指定值的项
事　件	**说　明**
DataBound	在 DropDownList 控件绑定到数据源后发生
SelectedIndexChanged	当列表控件的选定项改变并发回服务器时发生

5.5.2　使用 DropDownList 控件分类显示新闻

在任务 5.4 中提到了新闻文章管理页面有搜索新闻的功能，搜索条件可以有多种，其中一种就是可按新闻类别进行搜索。可以这样来设计用户界面，在页面上放置一个下拉列表供用户选择新闻类别，单击“查询”按钮，可在 GridView 控件提供的新闻列表中显示相应的查询结果。在任务 5.4 中，已经完成了 GetNewsCategories()和 GetNewsPartFieldsByConditions()两个方法的数据访问层及业务逻辑层的实现工作，前者用于获取新闻类别对象集合，后者用于根据查询条件、排序字段、排序方向返回包含部分字段的新闻对象集合。所以，现在只需对表示层做一些改进工作就行了，表示层的工作分为以下两步来完成。

1. 使用 DropDownList 控件显示新闻类别

基本步骤如下：

（1）添加数据源控件。打开任务 5.4 中已经创建的页面 NewsManager.aspx 并切换到“设计”视图，向页面拖放一个 ObjectDataSource 控件，设置其 ID 属性值为 odsNewsCategories。在配置数据源时，指定选择数据的方法是业务逻辑层 NewsCategoryManager 类的 GetNewsCategories()方法，用于获取新闻类别数据。

（2）添加 DropDownList 控件。向设计窗体拖放一个 DropDownList 控件，设置其 ID 属性值为 ddlNewsCategorySearch。单击 DropDownList 控件右上角的小三角按钮，然后在打开的下拉列表中选择“选择数据源”选项，弹出“数据源配置向导”对话框，在“选择数据源”下拉列表中选择 odsNewsCategories 选项，在“选择要在 DropDownList 控件中显示的数据字段”下拉列表中选择 Name 选项，在“为 DropDownList 的值选择数据字段”下拉列表中选择 Id 选项，如图 5-35 所示。或者将 ddlNewsCategorySearch 的 DataSourceID 属性值设置为 odsNewsCategories，DataTextField 属性值设置为 Name，DataValueField 属性值设置为 Id 完成相同功能。这样，就实现了 DropDownList 控件的数据绑定。运行程序，可看到新闻类别表(NewsCategories)中的数据已经显示在 DropDownList 控件上了。

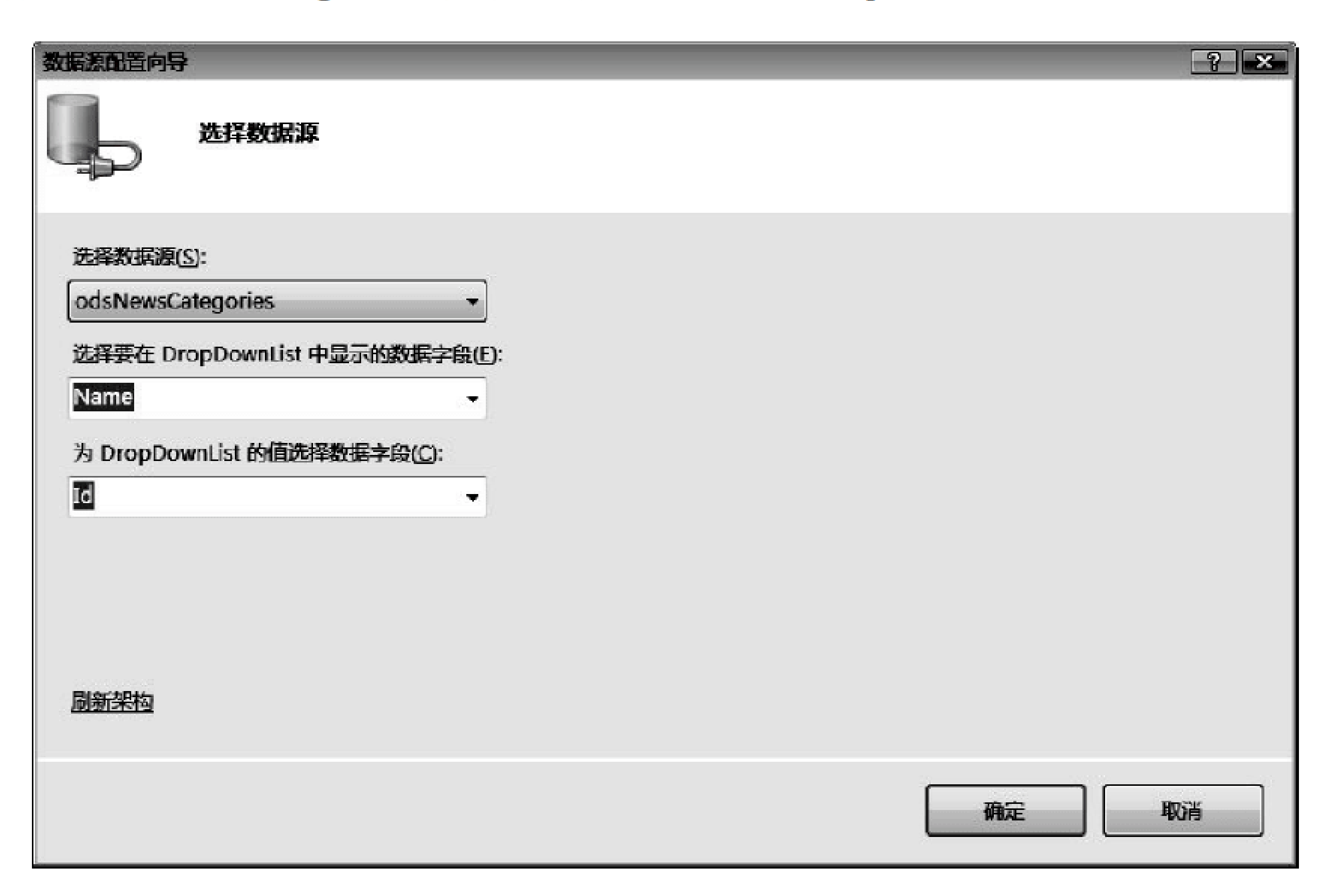


图 5-35 DropDownList 控件选择数据源

（3）为了给下拉列表再增加一个“全部”选项，可在 NewsManager.aspx.cs 中编写 ddlNewsCategorySearch_DataBound 事件代码进行动态绑定。页面运行后的新闻类别下拉列表如图 5-36 所示。



图 5-36 用 DropDownList 控件显示新闻类别列表运行界面

2. 构造复合查询条件实现新闻查询

基本步骤如下：

（1）设置查询输入控件。设计如图 5-37 所示的新闻文章

管理页界面，新闻的查询方式分为可按类别、标题、作者、起止日期等条件进行组合查询。其中，类别用前面已经完成的 DropDownList 控件选择，标题、作者用 TextBox 控件输入，起止日期用 TextBox 控件结合第三方控件 My97DatePicker 输入。My97DatePicker 的使用可参考任务 4.2。将下载的整个 My97DatePicker 文件夹放到 Web 项目根目录中，在页面中的引用方式参见下面的 HTML 代码。

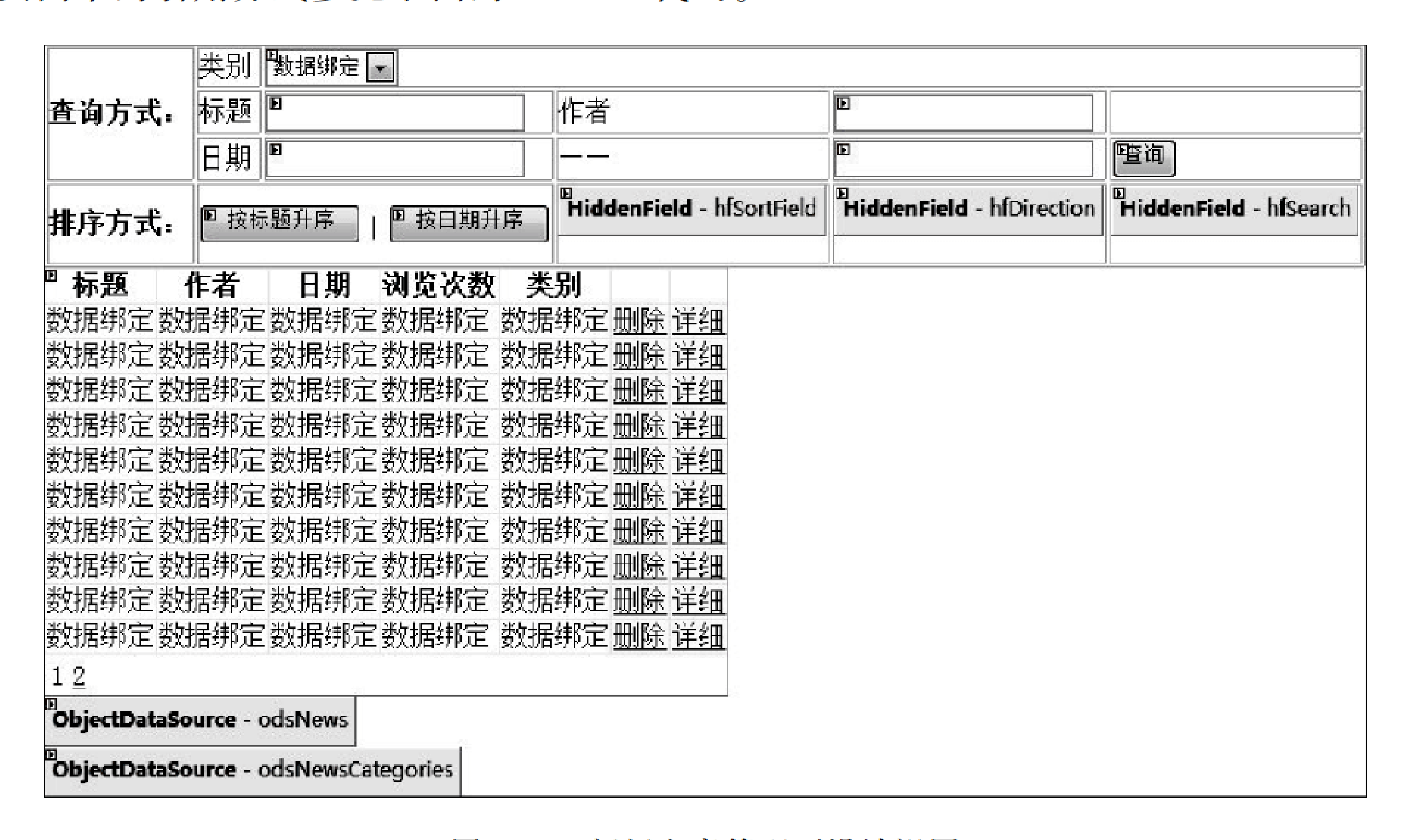


图 5-37　新闻文章管理页设计视图

(2) 编写查询事件。在“查询”按钮 btnSearch 的 Click 事件中编写代码构造复合查询条件，并以字符串形式保存在隐藏域控件 hfSearch 中。

改进后的新闻文章管理页 NewsManager.aspx 的关键 HTML 代码如下所示：

```
<%@ Page Language="C#" MasterPageFile="~/Admin/Admin.Master" AutoEventWireup=
"true" CodeBehind="NewsManager.aspx.cs" Inherits="Web.Admin.NewsManager" Title="文章
管理" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <script language="javascript" type="text/javascript" src="../My97DatePicker/
WdatePicker.js"></script>
    <table>
        <tr><th rowspan="3">查询方式：</th>
            <td>类别</td>
            <td colspan="4">
                <asp:DropDownList ID="ddlNewsCategorySearch" runat="server" DataSourceID=
"odsNewsCategories" DataTextField="Name" DataValueField="Id" OnDataBound="ddlNewsCategory-
Search_DataBound"></asp:DropDownList></td></tr>
        <tr><td>标题</td>
            <td><asp:TextBox ID="txtTitleSearch" runat="server"></asp:TextBox></td>
            <td>作者</td>
            <td><asp:TextBox ID="txtAuthorSearch" runat="server"></asp:TextBox></td>
            <td></td></tr>
```

```
        <tr><td>日期</td>
            <td><asp:TextBox ID="txtStartTime" runat="server" CssClass="Wdate"
onClick="WdatePicker({dateFmt:'yyyy-MM-dd HH:mm:ss'})"></asp:TextBox></td>
            <td>——</td>
            <td><asp:TextBox ID="txtEndTime" runat="server" CssClass="Wdate" onClick=
"WdatePicker({dateFmt:'yyyy-MM-dd HH:mm:ss'})"></asp:TextBox></td>
            <td><asp:Button ID="btnSearch" runat="server" Text="查询" OnClick=
"btnSearch_Click" /></td></tr>
        <tr><th>排序方式：</th>
            <td colspan="2"><asp:Button ID="btnTitleSort" runat="server" Text="按标
题升序" OnClick="btnTitleSort_Click" />|<asp:Button ID="btnPubDateSort" runat=
"server" Text="按日期升序" OnClick="btnPubDateSort_Click" /></td>
            <td><asp:HiddenField ID="hfSortField" runat="server" Value="Id" /></td>
            <td><asp:HiddenField ID="hfDirection" runat="server" Value="DESC" /></td>
            <td><asp:HiddenField ID="hfSearch" runat="server" Value="1=1" /></td>
</tr>
    </table>
    <div id="admin_div">
        <asp:GridView ID="gvNews" runat="server" AutoGenerateColumns="False" DataSourceID=
"odsNews" DataKeyNames="Id" AllowPaging="True">
            <%--此处代码省略,参见任务 5.4--%>
        </asp:GridView>
        <asp:ObjectDataSource ID="odsNews" runat="server" DataObjectTypeName="NewsModels.
News" DeleteMethod="DeleteNews" SelectMethod="GetNewsPartFieldsByConditions" TypeName=
"NewsBLL.NewsManager">
            <%--此处代码省略,参见任务 5.4--%>
        </asp:ObjectDataSource>
        <asp:ObjectDataSource ID="odsNewsCategories" runat="server" SelectMethod=
"GetNewsCategories" TypeName="NewsBLL.NewsCategoryManager"></asp:ObjectDataSource>
    </div>
</asp:Content>
```

NewsManager.aspx.cs 的代码如下所示：

```
...
namespace Web.Admin
{
    public partial class NewsManager : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                ViewState["TitleClick"] = "0";
                ViewState["PubDateClick"] = "0";
            }
        }
        protected void btnSearch_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
```

```
        {
            if (txtStartTime.Text != "")
            {
                try
                {
                    Convert.ToDateTime(txtStartTime.Text);
                }
                catch (Exception)
                {
                    Common.Message.RegScript(this, "日期格式错误!");
                    return;
                }
            }
            if (txtEndTime.Text != "")
            {
                try
                {
                    Convert.ToDateTime(txtEndTime.Text);
                }
                catch (Exception)
                {
                    Common.Message.RegScript(this, "日期格式错误!");
                    return;
                }
            }
            if (ddlNewsCategorySearch.SelectedIndex == 0)
            {
                if (txtStartTime.Text == "")
                {
                    if (txtEndTime.Text == "")
                    {
                        hfSearch.Value = "Title LIKE '%" + txtTitleSearch.Text.
Trim() + "%' AND Author LIKE '%" + txtAuthorSearch.Text.Trim() + "%'";
                    }
                    else
                    {
                        hfSearch.Value = "Title LIKE '%" + txtTitleSearch.Text.Trim() +
"%' AND Author LIKE '%" + txtAuthorSearch.Text.Trim() + "%' AND PubDate <= '" + txtEndTime.
Text.Trim() + "'";
                    }
                }
                else
                {
                    if (txtEndTime.Text == "")
                    {
                        hfSearch.Value = "Title LIKE '%" + txtTitleSearch.Text.Trim() +
"%' AND Author LIKE '%" + txtAuthorSearch.Text.Trim() + "%' AND PubDate >= '" + txtStartTime.
Text.Trim() + "'";
                    }
```

```
                        else
                        {
                            hfSearch.Value = "Title LIKE '%" + txtTitleSearch.Text.
Trim() + "%' AND Author LIKE '%" + txtAuthorSearch.Text.Trim() + "%' AND PubDate>='" +
txtStartTime.Text.Trim() + "' AND PubDate<='" + txtEndTime.Text.Trim() + "'";
                        }
                    }
                }
                else
                {
                    if (txtStartTime.Text == "")
                    {
                        if (txtEndTime.Text == "")
                        {
                            hfSearch.Value = "NewsCategoryId=" + ddlNewsCategorySearch.
SelectedValue + " AND Title LIKE '%" + txtTitleSearch.Text.Trim() + "%' AND Author LIKE '%" +
txtAuthorSearch.Text.Trim() + "%'";
                        }
                        else
                        {
                            hfSearch.Value = "NewsCategoryId=" + ddlNewsCategorySearch.
SelectedValue + " AND Title LIKE '%" + txtTitleSearch.Text.Trim() + "%' AND Author LIKE '%" +
txtAuthorSearch.Text.Trim() + "%' AND PubDate<='" + txtEndTime.Text.Trim() + "'";
                        }
                    }
                    else
                    {
                        if (txtEndTime.Text == "")
                        {
                            hfSearch.Value = "NewsCategoryId=" + ddlNewsCategorySearch.
SelectedValue + " AND Title LIKE '%" + txtTitleSearch.Text.Trim() + "%' AND Author LIKE '%" +
txtAuthorSearch.Text.Trim() + "%' AND PubDate>='" + txtStartTime.Text.Trim() + "'";
                        }
                        else
                        {
                            hfSearch.Value = "NewsCategoryId=" + ddlNewsCategorySearch.
SelectedValue + " AND Title LIKE '%" + txtTitleSearch.Text.Trim() + "%' AND Author LIKE '%" +
txtAuthorSearch.Text.Trim() + "%' AND PubDate>='" + txtStartTime.Text.Trim() + "' AND
PubDate<='" + txtEndTime.Text.Trim() + "'";
                        }
                    }
                }
            }
        }
        protected void ddlNewsCategorySearch_DataBound(object sender, EventArgs e)
        {
            ListItem item = new ListItem("全部", "-1");
            ddlNewsCategorySearch.Items.Insert(0, item);
        }
```

```
        protected void btnTitleSort_Click(object sender, EventArgs e)
        {
            hfSortField.Value = "Title";
            if (ViewState["TitleClick"].ToString() == "0")
            {
                ViewState["TitleClick"] = "1";
                hfDirection.Value = "ASC";
                btnTitleSort.Text = "按标题降序";
            }
            else if (ViewState["TitleClick"].ToString() == "1")
            {
                ViewState["TitleClick"] = "0";
                hfDirection.Value = "DESC";
                btnTitleSort.Text = "按标题升序";
            }
        }
        protected void btnPubDateSort_Click(object sender, EventArgs e)
        {
            hfSortField.Value = "PubDate";
            if (ViewState["PubDateClick"].ToString() == "0")
            {
                ViewState["PubDateClick"] = "1";
                hfDirection.Value = "ASC";
                btnPubDateSort.Text = "按日期降序";
            }
            else if (ViewState["PubDateClick"].ToString() == "1")
            {
                ViewState["PubDateClick"] = "0";
                hfDirection.Value = "DESC";
                btnPubDateSort.Text = "按日期升序";
            }
        }
    }
}
```

新闻文章管理页最终运行效果如图 5-38 所示。

5.5.3　小结

（1）DropDownList 控件可结合数据源控件进行数据绑定。其两个重要属性：DataTextField 控件用于获取或设置为列表项提供文本内容的数据源字段；DataValueField 控件用于获取或设置为列表项提供值的数据源字段。

（2）构造模糊查询的格式为：SELECT 字段列表 FROM 表名 WHERE 字段名 LIKE '字符串'。

5.5.4　思考与练习

1．如何通过程序控制动态地为 DropDownList 控件增加列表项？



图 5-38　新闻文章管理页最终运行效果

2. 完成管理员后台用户管理页的搜索和排序功能。

3. 完成管理员后台留言管理页的搜索和排序功能。

任务 5.6　使用 DetailsView 控件实现新闻详细显示

任务目标

（1）会用 DetailsView 控件显示、插入、编辑数据。

（2）会用 FCKeditor 控件实现 HTML 文本格式设置、图片上传。

5.6.1　DetailsView 控件简介

顾名思义，DetailsView 控件是用于查看细节信息的控件，通常用在主/从方案里。在这种方案下，主控件（如 GridView 控件）中的所选记录决定了 DetailsView 控件显示的记录。

DetailsView 控件与 GridView 控件一样都继承于 CompositeDataBoundControl 类，因此它们具有很多共同的属性，但也有所不同。

DetailsView 控件与 GridView 控件都以表格方式显示数据。GridView 控件是面向整个记录集合的，而 DetailsView 控件则是面向单条记录的。在 DetailsView 控件的界面中每次只能显示一条记录，而且内容按照垂直方向排列（即表格的每一行显示记录的一个字段）。

在查询中若出现多条符合条件的记录时，DetailsView 控件将以分页显示的方法处理。

GridView 控件没有提供数据插入功能，但 DetailsView 控件却弥补了这个不足。DetailsView 控件不仅可以显示与它相关联数据源中的一条记录，而且还可以编辑、插入或删除记录。DetailsView 控件的常用属性与事件如表 5-16 所示。

表 5-16　DetailsView 控件的常用属性与事件

属　　性	说　　明
AutoGenerateRows	获取或设置一个值，该值指示对应于数据源中每个字段的行字段是否自动生成并在 DetailsView 控件中显示
CurrentMode	获取 DetailsView 控件的当前数据输入模式
DefaultMode	获取或设置 DetailsView 控件的默认数据输入模式。该属性为枚举值，包括 ReadOnly（显示）、Edit（编辑）、Insert（插入）
DataKeyNames	获取或设置一个数组，该数组包含数据源的键字段的名称
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DataSourceID	获取或设置控件的 ID，数据绑定控件从该控件中检索其数据项列表
事　　件	**说　　明**
DataBound	在 DetailsView 控件完成到数据源的绑定后发生
ItemDeleting	单击 DetailsView 控件中的“删除”按钮时，在删除记录之前发生
ItemDeleted	单击 DetailsView 控件中的“删除”按钮时，在删除记录之后发生
ItemInserting	单击 DetailsView 控件中的“插入”按钮时，在插入记录之前发生
ItemInserted	单击 DetailsView 控件中的“插入”按钮时，在插入记录之后发生
ItemUpdating	单击 DetailsView 控件中的“更新”按钮时，在更新记录之前发生
ItemUpdated	单击 DetailsView 控件中的“更新”按钮时，在更新记录之后发生

5.6.2　新闻详细显示数据访问层与业务逻辑层的实现

当新闻详细显示时，数据访问层与业务逻辑层要能实现根据新闻 Id 值从新闻表（News）里查找相应的新闻记录，并返回一个新闻对象。

1. 新闻详细显示数据访问层的实现

在 NewsDAL 项目的 NewsService 类中增加如下代码：

```
/// <summary>
/// 根据 Id 查询新闻
/// </summary>
/// <param name = "id">新闻 Id</param>
/// <returns>新闻对象</returns>
public static News GetNewsById(int id)
{
    using (SqlConnection cn = new SqlConnection(connectionString))
    {
        cn.Open();
        SqlCommand cm = new SqlCommand();
        cm.Connection = cn;
        string sql = "SELECT * FROM News WHERE Id = @Id";
        cm.CommandText = sql;
```

```
            cm.Parameters.AddWithValue("@Id", id);
            SqlDataReader dr = cm.ExecuteReader();
            if (dr.Read())
            {
                News news = new News();
                news.Id = (int)dr["Id"];
                news.Title = (string)dr["Title"];
                news.Author = (string)dr["Author"];
                news.PubDate = (DateTime)dr["PubDate"];
                news.Contents = (string)dr["Contents"];
                news.Clicks = (int)dr["Clicks"];
                news.NewsCategoryId = (int)dr["NewsCategoryId"];
                news. NewsCategory = NewsCategoryService. GetNewsCategoryById ((int) dr
["NewsCategoryId"]);
                dr.Close();
                return news;
            }
            else
            {
                dr.Close();
                return null;
            }
        }
}
```

2. 新闻详细显示业务逻辑层的实现

在 NewsBLL 项目的 NewsManager 类中增加如下代码：

```
public static News GetNewsById(int id)
{
    return NewsService.GetNewsById(id);
}
```

5.6.3 使用 DetailsView 控件实现管理员后台新闻详细显示

在任务 5.4 的新闻文章管理页 NewsManager.aspx 中，使用 GridView 控件实现了新闻列表显示。列表中有个显示为“详细”的 HyperLinkField 列，它是用于指向新闻详细页的链接。单击“详细”超链接，可跳转到新闻详细页，同时还将当前新闻记录的 Id 值随 URL 地址一起传递过去。下面来编写新闻详细页面。由于 DetailsView 控件可以显示数据库中单条记录的详细信息，所以可使用该控件完成新闻详细页面。在新闻详细页中，先根据由 NewsManager.aspx 页传递过来的新闻 Id 值从新闻表(News)里查找相应的新闻记录，再用 DetailsView 控件显示该新闻记录的所有字段信息。

在 Web 项目的 Admin 文件夹下，新建页面文件 EditNews.aspx。

使用 DetailsView 控件显示新闻详细信息的基本步骤如下。

(1) 添加数据源控件。切换到“设计”视图，向页面拖放一个 ObjectDataSource 控件，设置其 ID 属性值为 odsNews。在配置数据源时，指定选择数据的方法是业务逻辑层

NewsManager 类的 GetNewsById()方法,用于获取新闻详细数据。其查询参数值来自查询字符串,设置方式如图 5-39 所示。

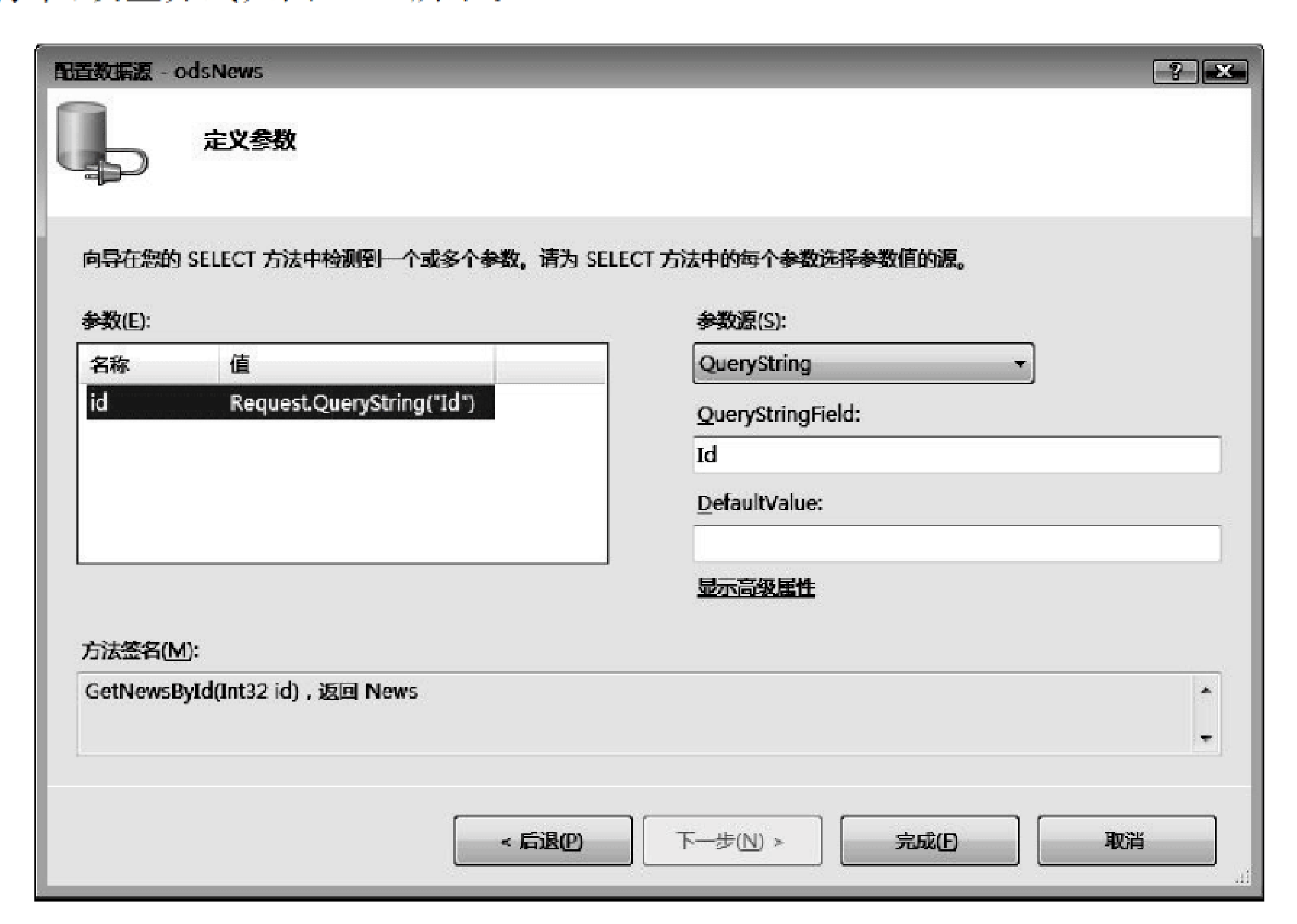


图 5-39　为新闻详细显示的 SELECT 方法定义参数

(2) 添加 DetailsView 控件。将工具箱"数据"选项卡中的 DetailsView 控件拖放到页面中,设置 DetailsView 控件的 ID 属性值为 dvNews。单击 DetailsView 控件右上角的小三角按钮打开 DetailsView 任务菜单,在"选择数据源"下拉列表中选择数据源 odsNews,将数据源绑定到 DetailsView 控件。在 DatailsView 任务菜单中,选择"编辑字段",打开"字段"对话框,为需要显示的数据源中的字段 Id、Title、Author、PubDate、Contents、Clicks、NewsCategory. Name 手动添加 7 个 BoundField 列,并设置各个 BoundField 列的 DataField 属性为对应的字段名,设置 HeaderText 属性为中文标题。需要注意的是,NewsCategory 是外键对象,所以必须将这个 BoundField 列转换成 TemplateField 列。这样就实现了 DetailsView 控件的数据绑定。运行界面如图 5-40 所示,可看到某条新闻的详细信息已经显示在 DetailsView 控件上了。以下是生成的 HTML 关键代码:

```
<asp:ObjectDataSource ID = "odsNews" runat = "server" SelectMethod = "GetNewsById" TypeName =
"NewsBLL.NewsManager" DataObjectTypeName = "NewsModels.News">
    <SelectParameters>
        <asp:QueryStringParameter Name = "id" QueryStringField = "Id" Type = "Int32" />
    </SelectParameters>
</asp:ObjectDataSource>
<asp:DetailsView ID = "dvNews" runat = "server" AutoGenerateRows = "False" DataSourceID =
"odsNews">
    <Fields>
```

```
        <asp:BoundField DataField="Id" HeaderText="Id" />
        <asp:BoundField DataField="Title" HeaderText="标题" />
        <asp:BoundField DataField="Author" HeaderText="作者" />
        <asp:BoundField DataField="PubDate" HeaderText="日期" />
        <asp:BoundField DataField="Contents" HeaderText="内容" />
        <asp:BoundField DataField="Clicks" HeaderText="浏览次数" />
        <asp:TemplateField HeaderText="类别">
            <EditItemTemplate>
                <asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind
("NewsCategory.Name") %>'></asp:TextBox>
            </EditItemTemplate>
            <InsertItemTemplate>
                <asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind
("NewsCategory.Name") %>'></asp:TextBox>
            </InsertItemTemplate>
            <ItemTemplate>
                <asp:Label ID="Label1" runat="server" Text='<%# Bind("NewsCategory.
Name") %>'></asp:Label>
            </ItemTemplate>
        </asp:TemplateField>
    </Fields>
</asp:DetailsView>
```

Id	14
标题	以旧换新拉动新车消费
作者	吴晓
日期	2009/7/2 15:00:00
内容	全国现有黄标车1800万辆待换。汽车以旧换新不仅是拉动消费的动力，也是为了环境保护、节约资源、节约能源、带动就业等多方面的目的，所以是一个长期的战略任务，发展空间很大。
浏览次数	0
类别	财经

图 5-40　用 DetailsView 控件显示新闻详细信息运行效果

5.6.4　新闻编辑、添加数据访问层与业务逻辑层的实现

1. 新闻编辑、添加数据访问层的实现

在 NewsDAL 项目的 NewsService 类中增加如下代码：

```
/// <summary>
/// 添加新闻
/// </summary>
/// <param name="news">新闻对象</param>
public static void AddNews(News news)
{
    using (SqlConnection cn = new SqlConnection(connectionString))
    {
        cn.Open();
        SqlCommand cm = new SqlCommand();
        cm.Connection = cn;
        string sql =
        "INSERT News (Title, Author, PubDate, Contents, Clicks, NewsCategoryId)" +
```

```
            "VALUES (@Title, @Author, @PubDate, @Contents, @Clicks, @NewsCategoryId)";
            cm.CommandText = sql;
            cm.Parameters.AddWithValue("@Title", news.Title);
            cm.Parameters.AddWithValue("@Author", news.Author);
            cm.Parameters.AddWithValue("@PubDate", news.PubDate);
            cm.Parameters.AddWithValue("@Contents", news.Contents);
            cm.Parameters.AddWithValue("@Clicks", news.Clicks);
            cm.Parameters.AddWithValue("@NewsCategoryId", news.NewsCategoryId);
            cm.ExecuteNonQuery();
        }
    }

    /// <summary>
    /// 修改新闻
    /// </summary>
    /// <param name = "news">新闻对象</param>
    public static void ModifyNews(News news)
    {
        using (SqlConnection cn = new SqlConnection(connectionString))
        {
            cn.Open();
            SqlCommand cm = new SqlCommand();
            cm.Connection = cn;
            string sql =
            "UPDATE News " +
            "SET " +
                "Title = @Title, " +
                "Author = @Author, " +
                "PubDate = @PubDate, " +
                "Contents = @Contents, " +
                "Clicks = @Clicks, " +
                "NewsCategoryId = @NewsCategoryId " +
            "WHERE Id = @Id";
            cm.CommandText = sql;
            cm.Parameters.AddWithValue("@Id", news.Id);
            cm.Parameters.AddWithValue("@Title", news.Title);
            cm.Parameters.AddWithValue("@Author", news.Author);
            cm.Parameters.AddWithValue("@PubDate", news.PubDate);
            cm.Parameters.AddWithValue("@Contents", news.Contents);
            cm.Parameters.AddWithValue("@Clicks", news.Clicks);
            cm.Parameters.AddWithValue("@NewsCategoryId", news.NewsCategoryId);
            cm.ExecuteNonQuery();
        }
    }
```

2. 新闻编辑、添加业务逻辑层的实现

在 NewsBLL 项目的 NewsManager 类中增加如下代码：

```
public static void AddNews(News news)
{
```

```
        if (news.Author == null)
            news.Author = "";
        news.PubDate = DateTime.Now;
        news.Clicks = 0;
        NewsService.AddNews(news);
    }
    public static void ModifyNews(News news)
    {
        if (news.Author == null)
            news.Author = "";
        if (news.PubDate == DateTime.MinValue)
            news.PubDate = DateTime.Now;
        NewsService.ModifyNews(news);
    }
```

在上面代码中，对新闻编辑、添加时的一些字段默认值做了处理。比如添加一条新闻时，新闻日期取当前系统日期，新闻浏览次数设为0，作者名为空时用空字符串代替。

5.6.5 使用 DetailsView 控件实现新闻编辑、添加

DetailsView 控件经常用于编辑或插入记录。所以可用 DetailsView 控件来实现新闻系统管理员后台的新闻编辑与添加。在编辑与添加操作时，应注意考虑以下几点。

(1) 新闻编号(Id)、新闻浏览次数(Clicks)不允许修改，需要设置成只读模式。

(2) 新闻标题(Title)需要进行非空验证，可用 DetailsView 模板列结合验证控件实现。

(3) 在新闻日期(PubDate)和新闻内容(Contents)输入时分别要用到第三方控件 My97DatePicker 和 FCKeditor，所以这两列要转换成模板列。FCKeditor 是一款非常优秀的 HTML 在线文本编辑器，功能完善，具有开放的功能接口，是目前网络上最流行的编辑器之一，被很多网站采用。在输入新闻内容时，可用它进行文本格式设置、上传新闻图片等。

(4) 新闻类别(NewsCategory)输入需要以下拉列表的形式进行选择，在前面已设成模板列。注意 NewsCategory 作为一个对象的处理方式，以及进入编辑模式后对下拉列表初始选定项的处理，它应该为当前记录的新闻类别。

下面对页面文件 EditNews.aspx 中的 DetailsView 控件再做修改。

使用 DetailsView 控件编辑、添加新闻详细信息的基本步骤如下。

(1) 为数据源控件设置更新和插入数据方法。切换到“设计”视图，修改 odsNews 的数据源配置。在如图 5-41 所示的对话框中，打开 UPDATE 选项卡选择 ModifyNews(News news)方法，打开 DELETE 选项卡选择 AddNews(News news)方法。

(2) 将某些绑定列转换成模板列。在 DetailsView 任务菜单中，选择“编辑字段”命令，打开“字段”对话框，将除新闻作者(Author)之外的 5 个 BoundField 列转换成 TemplateField 列，NewsCategory 已是 TemplateField 列，无须再次转换。

(3) 设置某些列为只读模式。将 Id、Clicks 字段的 EditItemTemplate、InsertItemTemplate 项及 PubDate 字段的 InsertItemTemplate 项中自动生成的 TextBox 控件删除，替换成 Label 控件，并为每个 Label 控件的 Text 属性设置绑定表达式。以设置 Id 字段为例，如



图 5-41　定义数据方法——配置 UPDATE 操作

图 5-42 所示，单击 EditItemTemplate 中 Label 控件右上角的小三角按钮，在弹出的下拉列表中选择“编辑 DataBindings”选项，在打开的 Label6 DataBindings 对话框中，为 Text 属性设置绑定表达式 Bind("Id")。



图 5-42　为 EditItemTemplate 中 Label 控件的 Text 属性设置绑定字段

（4）为某些列引用第三方控件。在 PubDate 字段的 EditItemTemplate 项中引用 My97DatePicker 控件，将 Contents 字段的 EditItemTemplate、InsertItemTemplate 项中自动生成的 TextBox 控件替换成 FCKeditor 控件。

(5) 为某些列设置非空验证。在 Title 字段的 EditItemTemplate、InsertItemTemplate 项各设置一个 RequiredFieldValidator 控件用于新闻标题输入框的非空验证，在 Contents 字段的 EditItemTemplate、InsertItemTemplate 项各设置一个 CustomValidator 控件及 JavaScript 脚本用于新闻内容输入框的非空验证。

(6) 处理对象列。向页面添加一个 ID 名为 odsNewsCategories 的 ObjectDataSource 控件用于获取新闻类别数据，在 NewsCategory 字段的 EditItemTemplate、InsertItemTemplate 项各设置一个 DropDownList 提供选择新闻类别，用 HiddenField 控件保存编辑模式下 DropDownList 的初始选定项的值。

(7) 启用编辑和插入。在如图 5-43 所示的 DetailsView 任务菜单中选中"启用编辑"和"启用插入"复选框。

图 5-43　DetailsView 任务菜单

EditNews.aspx 实现代码如下：

```
<%@ Page Language = "C#" MasterPageFile = "~/Admin/Admin.Master" AutoEventWireup =
"true" CodeBehind = "EditNews.aspx.cs" Inherits = "Web.EditNews" Title = "修改|添加文章" %>
<%@ Register Assembly = "FredCK.FCKeditorV2" Namespace = "FredCK.FCKeditorV2" TagPrefix =
"FCKeditorV2" %>
<asp:Content ID = "Content1" ContentPlaceHolderID = "ContentPlaceHolder1" runat = "server">
    <script language = "javascript" type = "text/javascript" src = "../My97DatePicker/
WdatePicker.js"></script>
    <script language = "javascript" type = "text/javascript">
    var oEditer;
    function CustomValidate(source, arguments)
    {
        var value = oEditer.GetXHTML(true);
        if(value == "")
        {
            arguments.IsValid = false;
        }
        else
        {
            arguments.IsValid = true;
        }
    }
    function FCKeditor_OnComplete(editorInstance)
    {
        oEditer = editorInstance;
    }
    </script>
    <asp:DetailsView ID = "dvNews" runat = "server" AutoGenerateRows = "False" DataSourceID =
"odsNews" OnDataBound = "dvNews_DataBound" OnItemUpdating = "dvNews_ItemUpdating"
OnItemInserted = "dvNews_ItemInserted">
        <Fields>
```

```
                <asp:TemplateField HeaderText = "Id">
                    <EditItemTemplate>
                        <asp:Label ID = "Label6" runat = "server" Text = '<% # Bind("Id") %>'>
</asp:Label>
                    </EditItemTemplate>
                    <InsertItemTemplate>
                        <asp:Label ID = "Label7" runat = "server" Text = '<% # Bind("Id") %>'>
</asp:Label>
                    </InsertItemTemplate>
                    <ItemTemplate>
                        <asp:Label ID = "Label5" runat = "server" Text = '<% # Bind("Id") %>'>
</asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:TemplateField HeaderText = "标题">
                    <EditItemTemplate>
                        <asp:TextBox ID = "txtTitle1" runat = "server" Text = '<% # Bind("Title")
%>'></asp:TextBox>
                         <asp:RequiredFieldValidator ID = "RequiredFieldValidator1" runat =
"server" ErrorMessage = "标题不能为空" ControlToValidate = "txtTitle1"></asp:RequiredField-
Validator>
                    </EditItemTemplate>
                    <InsertItemTemplate>
                        <asp:TextBox ID = "txtTitle2" runat = "server" Text = '<% # Bind("Title")
%>'></asp:TextBox>
                        <asp: RequiredFieldValidator  ID  =  " RequiredFieldValidator2 "  runat  =
"server" ErrorMessage = "标题不能为空" ControlToValidate = "txtTitle2"></asp:RequiredField-
Validator>
                    </InsertItemTemplate>
                    <ItemTemplate>
                        <asp:Label ID = "Label3" runat = "server" Text = '<% # Bind("Title") %>'>
</asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:BoundField DataField = "Author" HeaderText = "作者" />
                <asp:TemplateField HeaderText = "日期">
                    <EditItemTemplate>
                        <asp:TextBox  ID  =  " txtPubDate"  runat  =  " server"  Text  =  '<%  #  Bind
("PubDate") %>' CssClass = "Wdate" onClick = "WdatePicker({dateFmt:'yyyy - MM - dd HH:mm:ss'})"></
asp:TextBox>
                    </EditItemTemplate>
                    <InsertItemTemplate>
                        <asp:Label  ID  =  " lblPubDate"  runat  =  " server"  Text  =  '<%  #  System.
DateTime.Now %>'></asp:Label>
                    </InsertItemTemplate>
                    <ItemTemplate>
                        <asp:Label  ID  =  " Label4 "  runat  =  " server "  Text  =  '<%  #  Bind
("PubDate") %>'></asp:Label>
                    </ItemTemplate>
```

```
                </asp:TemplateField>
                <asp:TemplateField HeaderText="内容">
                    <EditItemTemplate>
                        <FCKeditorV2:FCKeditor ID="FCKeditor1" runat="server" Value=
'<%# Bind("Contents") %>'></FCKeditorV2:FCKeditor>
                            <asp:CustomValidator ID="CustomValidator1" runat="server"
ErrorMessage="内容不能为空" ControlToValidate="FCKeditor1" ClientValidationFunction=
"CustomValidate" ValidateEmptyText="true"></asp:CustomValidator>
                    </EditItemTemplate>
                    <InsertItemTemplate>
                        <FCKeditorV2:FCKeditor ID="FCKeditor2" runat="server" Value=
'<%# Bind("Contents") %>'></FCKeditorV2:FCKeditor>
                        <asp:CustomValidator ID="CustomValidator2" runat="server"
ErrorMessage="内容不能为空" ControlToValidate="FCKeditor2" ClientValidationFunction=
"CustomValidate" ValidateEmptyText="true"></asp:CustomValidator>
                    </InsertItemTemplate>
                    <ItemTemplate>
                        <asp:Label ID="Label1" runat="server" Text='<%# Bind
("Contents") %>'></asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:TemplateField HeaderText="浏览次数">
                    <EditItemTemplate>
                        <asp:Label ID="Label9" runat="server" Text='<%# Bind("Clicks")
%>'></asp:Label>
                    </EditItemTemplate>
                    <InsertItemTemplate>
                        <asp:Label ID="Label10" runat="server" Text='<%# 0 %>'></asp:
Label>
                    </InsertItemTemplate>
                    <ItemTemplate>
                        <asp:Label ID="Label8" runat="server" Text='<%# Bind("Clicks")
%>'></asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:TemplateField HeaderText="类别">
                    <EditItemTemplate>
                        <asp:DropDownList ID="ddlNewsCategory1" runat="server" DataSourceID=
"odsNewsCategories" DataTextField="Name" DataValueField="Id">
                        </asp:DropDownList>
                        <asp:HiddenField ID="hfNewsCategoryId" runat="server" Value='<%
# Bind("NewsCategoryId") %>' />
                    </EditItemTemplate>
                    <InsertItemTemplate>
                        <asp:DropDownList ID="ddlNewsCategory2" runat="server" DataSourceID=
"odsNewsCategories" DataTextField="Name" DataValueField="Id" SelectedValue='<%# Bind
("NewsCategoryId") %>'>
                        </asp:DropDownList>
                    </InsertItemTemplate>
```

```
                    <ItemTemplate>
                        <asp:Label ID="Label2" runat="server" Text='<%# Bind("NewsCategory.
Name") %>'></asp:Label>
                    </ItemTemplate>
                </asp:TemplateField>
                <asp:CommandField ShowEditButton="True" ShowInsertButton="True" />
            </Fields>
        </asp:DetailsView>
        <asp:ObjectDataSource ID="odsNews" runat="server" SelectMethod="GetNewsById"
TypeName="NewsBLL.NewsManager" UpdateMethod="ModifyNews" DataObjectTypeName=
"NewsModels.News" InsertMethod="AddNews">
            <SelectParameters>
                <asp:QueryStringParameter Name="id" QueryStringField="Id" Type="Int32" />
            </SelectParameters>
        </asp:ObjectDataSource>
        <asp:ObjectDataSource ID="odsNewsCategories" runat="server" SelectMethod=
"GetNewsCategories" TypeName="NewsBLL.NewsCategoryManager">
        </asp:ObjectDataSource>
</asp:Content>
```

EditNews.aspx.cs 实现代码如下：

```
...
namespace Web
{
    public partial class EditNews : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }
        protected void dvNews_DataBound(object sender, EventArgs e)
        {
            if (dvNews.CurrentMode == DetailsViewMode.Edit)
            {
                DropDownList ddlNewsCategory1 = dvNews.FindControl("ddlNewsCategory1")
as DropDownList;
                HiddenField hfNewsCategoryId = dvNews.FindControl("hfNewsCategoryId")
as HiddenField;
                try
                {
                    ddlNewsCategory1.SelectedValue = hfNewsCategoryId.Value.Trim();
                    ddlNewsCategory1.Attributes["onchange"] = "javascript:document.
getElementById('" + hfNewsCategoryId.ClientID + "').value=this.value";
                }
                catch { };
            }
        }
        //更新新闻,数据源更新方法执行之前,增加参数
        protected void dvNews_ItemUpdating(object sender, DetailsViewUpdateEventArgs e)
```

```
        {
            if (Page.IsValid)
            {
                TextBox txtPubDate = dvNews.FindControl("txtPubDate") as TextBox;
                if (txtPubDate.Text != "")
                {
                    try
                    {
                        Convert.ToDateTime(txtPubDate.Text);
                    }
                    catch (Exception)
                    {
                        Common.Message.RegScript(this, "日期格式错误!");
                        return;
                    }
                }
                //获得新闻类别下拉列表的值
                DropDownList ddlNewsCategory1 = dvNews.FindControl("ddlNewsCategory1")
as DropDownList;
                //添加新闻类别 Id 的参数
                odsNews. UpdateParameters. Add ( " newsCategoryId ", ddlNewsCategory1.
SelectedValue);
            }
        }
        // 添加成功后,跳转到列表页
        protected void dvNews_ItemInserted(object sender, DetailsViewInsertedEventArgs e)
        {
            Response.Redirect("NewsManager.aspx");
        }
    }
}
```

管理员后台新闻详细页最终运行效果如图 5-44 所示。



图 5-44　管理员后台新闻详细页最终运行效果

管理员后台新闻详细页编辑模式最终运行效果如图 5-45 所示。

图 5-45　管理员后台新闻详细页编辑模式最终运行效果

管理员后台新闻详细页插入模式最终运行效果如图 5-46 所示。

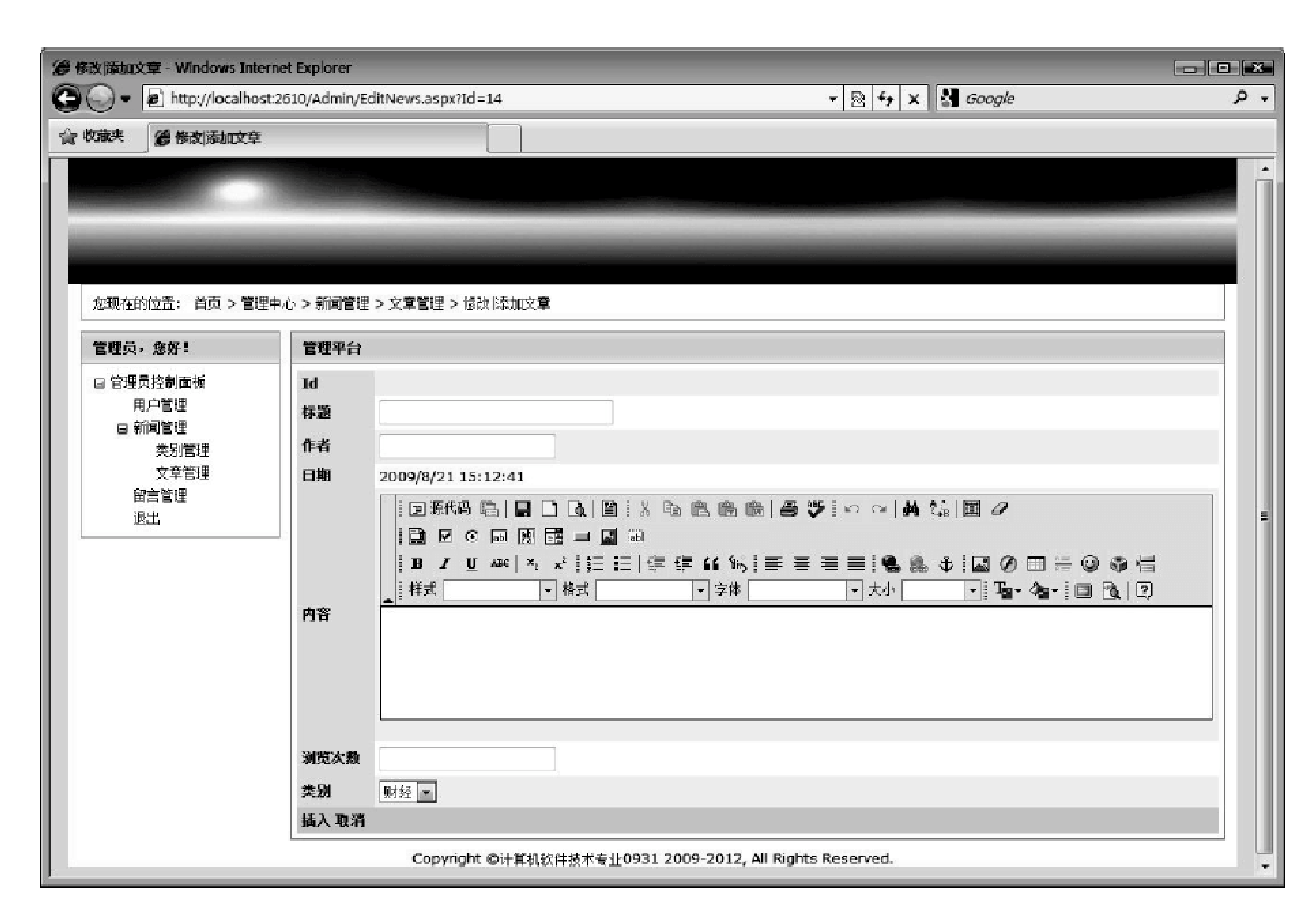


图 5-46　管理员后台新闻详细页插入模式最终运行效果

5.6.6 小结

(1) DetailsView 控件一次呈现一条表格形式的记录(即一条记录分多行显示,记录的一个字段占表格的一行),并提供翻阅多条记录以及插入、更新和删除记录的功能。它可与 GridView 控件结合使用,以用于主/从方案。DetailsView 控件的字段、模板等用法与 GridView 控件一致。

(2) 若要将某些 TemplateField 列设置为只读模式,可在该列的 EditItemTemplate、InsertItemTemplate 项中用 Label 控件绑定字段值。

(3) FCKeditor 是一款功能强大的开源在线文本编辑器,它支持 HTML 文本格式设置、文件上传等多种功能。

5.6.7 思考与练习

1. 用 DetailsView 控件实现管理员后台新闻类别的添加。
2. 用 DetailsView 控件实现管理员后台用户信息的编辑。

任务 5.7 使用 FormView 控件实现新闻详细显示

任务目标

(1) 会用 FormView 控件显示数据。
(2) 会用 Bind 和 Eval 两种形式的数据绑定表达式。

5.7.1 FormView 控件简介

FormView 控件与 DetailsView 控件类似,它一次呈现数据源中的一条记录,并提供翻阅多条记录以及插入、更新和删除记录的功能。不过,它们之间也存在差别:在 DetailsView 控件的表格布局中,数据记录的每个字段都各自显示为一行;而 FormView 控件则不指定用于显示记录的预定义布局。如果利用 FormView 控件显示内容,需要为其不同部分创建模板,以显示记录中的各个字段。该模板包含用于设置窗体布局的格式、控件和绑定表达式。所以使用 FormView 控件可以使控制数据显示更加灵活。FormView 控件的常用属性与事件如表 5-17 所示。

表 5-17 FormView 控件的常用属性与事件

属 性	说 明
AutoGenerateColumns	获取或设置一个值,该值指示是否为数据源中的每个字段自动创建绑定字段
AllowPaging	获取或设置一个值,该值指示是否启用分页功能
PageSize	获取或设置 GridView 控件在每页上所显示的记录数目
DataKeyNames	获取或设置一个数组,该数组包含了显示在 GridView 控件中的项的主键字段的名称

续表

属　性	说　明
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DataSourceID	获取或设置控件的 ID，数据绑定控件从该控件中检索其数据项列表
事　件	**说　明**
DataBound	在 GridView 控件完成到数据源的绑定后发生
RowDataBound	在 GridView 控件中的某个行被绑定到一个数据记录时发生
PageIndexChanging	单击“页导航”按钮时，在 GridView 控件执行分页操作之前发生
RowDeleting	单击 GridView 控件内某一行的“删除”按钮时，在 GridView 控件从数据源删除该行记录之前发生
RowDeleted	单击 GridView 控件内某一行的“删除”按钮时，在 GridView 控件从数据源删除该行记录之后发生
RowEditing	单击 GridView 控件内某一行的“编辑”按钮时，在 GridView 控件进入编辑模式之前发生
RowCancelingEdit	单击 GridView 控件内某一行的“编辑”按钮时，在 GridView 控件退出编辑模式之前发生
RowUpdating	单击 GridView 控件内某一行的“更新”按钮时，在 GridView 控件更新记录之前发生
RowUpdated	单击 GridView 控件内某一行的“更新”按钮时，在 GridView 控件更新记录之后发生

5.7.2　使用 FormView 控件实现前台新闻详细显示

与前台新闻详细页相关的数据访问层、业务逻辑层方法 GetNewsById()已在任务 5.6 中完成，下面只需实现前台新闻详细页的表示层。其做法类似管理员后台新闻详细页，只不过这次用 FormView 控件来显示新闻详细内容。

在 Web 项目的根目录下，新建页面文件 NewsDetail.aspx。

使用 FormView 控件显示新闻详细信息的基本步骤如下。

(1) 添加数据源控件。切换到“设计”视图，向页面拖放一个 ObjectDataSource 控件，设置其 ID 属性值为 odsNews。在配置数据源时，指定选择数据的方法是业务逻辑层 NewsManager 类的 GetNewsById()方法，用于获取新闻详细数据。其查询参数值来自查询字符串，设置方式如图 5-39 所示。

(2) 添加 FormView 控件。将工具箱“数据”选项卡中的 FormView 控件拖放到页面中，设置 FormView 控件的 ID 属性值为 fvNews。单击 FormView 控件右上角的小三角按钮打开 FormView 任务菜单，选择“编辑模板”命令，在 ItemTemplate 项里自由布局各字段的显示方式。如图 5-47 所示为在 ItemTemplate 项里加入一个3 行 3 列的表格布局，并设置各列的绑定字段。

数据绑定
作者：数据绑定 发表日期：数据绑定 浏览次数：数据绑定
数据绑定

图 5-47　用 FormView 控件显示新闻详细设计视图

NewsDetail.aspx 实现代码如下：

```
<%@ Page Language="C#" MasterPageFile="~/NewsPage.Master" AutoEventWireup="true"
```

```
CodeBehind = "NewsDetail.aspx.cs" Inherits = "Web.NewsDetail" Title = "新闻详细" %>
<asp:Content ID = "Content1" ContentPlaceHolderID = "ContentPlaceHolder1" runat = "server">
    <div id = "col">
        <div class = "box">
            <div class = "box_title">新闻详细</div>
            <div class = "box_text">
                <asp:FormView ID = "fvNews" runat = "server">
                    <ItemTemplate>
                        <table>
                            <tr>
                                <td colspan = "3"><%# Eval("Title") %></td>
                            </tr>
                            <tr>
                                <td>作者:<%# Eval("Author") %></td>
                                <td>发表日期:<%# Eval("PubDate") %></td>
                                <td>浏览次数:<%# Eval("Clicks") %></td>
                            </tr>
                            <tr>
                                <td colspan = "3"><%# Eval("Contents") %></td>
                            </tr>
                        </table>
                    </ItemTemplate>
                </asp:FormView>
                <asp:ObjectDataSource ID = " odsNews" runat = " server" SelectMethod =
"GetNewsById" TypeName = "NewsBLL.NewsManager">
                    <SelectParameters>
                        <asp:QueryStringParameter Name = " id" QueryStringField = " Id"
Type = "Int32" />
                    </SelectParameters>
                </asp:ObjectDataSource>
            </div>
        </div>
    </div>
</asp:Content>
```

NewsDetail.aspx.cs 实现代码如下：

```
...
using NewsModels;
using NewsBLL;
namespace Web
{
    public partial class NewsDetail : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                if (Request.QueryString["Id"] != null)
                {
```

```
                int id;
                try
                {
                    id = Convert.ToInt32(Request.QueryString["Id"]);
                    News news = NewsManager.GetNewsById(id);
                    news.Clicks ++ ;
                    NewsManager.ModifyNews(news);
                    fvNews.DataSource = odsNews;
                    fvNews.DataBind();
                }
                catch
                {
                    return;
                }
            }
            else
            {
                return;
            }
        }
    }
}
}
```

前台新闻详细页最终运行效果如图 5-48 所示。



图 5-48　前台新闻详细页最终运行效果

5.7.3　小结

（1）FormView 控件用于一次显示数据源中的一条记录。与 DetailsView 控件相似，只是显示的是用户自定义的模板，而不是行字段。使用 FormView 控件可以使控制数据显示更加灵活。

(2) 数据绑定表达式包含在＜％＃和％＞分隔符之内，并使用 Eval 或 Bind 方法。Eval 方法是静态(只读)方法，用于定义单向(只读)绑定。该方法采用数据字段的值作为参数并将其作为字符串返回，一般用在绑定时需要格式化字符串的情况下。Bind 方法支持读/写功能，用于定义双向(可更新)绑定。该方法可以检索数据绑定控件的值并将任何更改提交回数据库。当绑定的数据可以被修改时，还是要使用 Bind 方法。

5.7.4 思考与练习

1. GridView、DetailsView、FormView 这 3 个控件各有什么特点？用法上有什么联系和区别？各适用于什么场合？

2. 数据绑定表达式的写法有哪两种方式？各适用于什么场合？

第 6 章　深入数据库编程

任务 6.1　使用 DataList 列表显示新闻

任务目标

（1）会用 DataList 控件显示数据。

（2）会在页面上绑定显示经截取和过滤后的字符串。

6.1.1　DataList 控件简介

DataList 控件与 GridView 控件一样，可用来显示、编辑或删除表中的记录。但是，DataList 控件能以更自由的方式显示数据，如在一行中显示多条记录等。可在 DataList 提供的模板中定义数据显示布局，比如可以为项、交替项、选定项和编辑项创建模板，也可以使用标题、脚注和分隔符模板自定义 DataList 的整体外观。DataList 控件支持的模板如表 6-1 所示。

表 6-1　DataList 控件支持的模板

模 板 属 性	说　　明
ItemTemplate	为 DataList 中的项提供内容和布局所要求的模板
AlternatingItemTemplate	如果已定义，则为 DataList 中的交替项提供内容和布局。如果未定义，则使用 ItemTemplate
EditItemTemplate	如果已定义，则为 DataList 中当前编辑的项提供内容和布局。如果未定义，则使用 ItemTemplate
SelectedItemTemplate	如果已定义，则为 DataList 中当前选定项提供内容和布局。如果未定义，则使用 ItemTemplate
HeaderTemplate	如果已定义，则为 DataList 的页眉节提供内容和布局。如果未定义，将不显示页眉节
FooterTemplate	如果已定义，则为 DataList 的脚注部分提供内容和布局。如果未定义，将不显示脚注部分
SeparatorTemplate	如果已定义，则为 DataList 中各项之间的分隔符提供内容和布局。如果未定义，将不显示分隔符

6.1.2　新闻速览数据访问层与业务逻辑层的实现

DataList 控件的一大特点是能实现灵活复杂的页面布局，所以可以通过它来做新闻速览页的新闻列表显示。要完成这一功能，仍然先从数据访问层与业务逻辑层着手。新

闻速览需要取出新闻表所有字段信息，类似任务 5.4 中取新闻表(News)部分字段方法 GetNewsPartFieldsByConditions()的做法，在数据访问层与业务逻辑层中分别新增一个方法用于取新闻表全部字段。

在 NewsDAL 项目的 NewsService 类中，增加一个根据查询条件、排序字段、排序方向返回包含所有字段的新闻对象集合的方法。实现代码如下：

```
/// <summary>
/// 根据查询条件、排序字段、排序方向返回包含所有字段的新闻列表
/// </summary>
/// <param name = "conditions">查询条件</param>
/// <param name = "sortField">排序字段</param>
/// <param name = "direction">排序方向</param>
/// <returns>新闻对象集合</returns>
public static IList < News > GetNewsAllFieldsByConditions ( string conditions, string
sortField, string direction)
{
    string sql = "SELECT Id, Title, Author, PubDate, Contents, Clicks, NewsCategoryId FROM News";
    if (conditions.Trim().Length > 0)
    {
        sql + = " WHERE " + conditions;
    }
    if (sortField.Trim().Length > 0)
    {
        sql + = " ORDER BY " + sortField;
    }
    if (direction.Trim().Length > 0)
    {
        sql + = " " + direction;
    }
    return GetNewsBySql(sql);
}
```

在 NewsBLL 项目的 NewsManager.cs 类中，增加相应方法。实现代码如下：

```
public static IList < News > GetNewsAllFieldsByConditions ( string conditions, string
sortField, string direction)
{
    return NewsService.GetNewsAllFieldsByConditions(conditions, sortField, direction);
}
```

6.1.3 使用 DataList 控件实现新闻速览列表显示

新闻速览页让用户能快速地浏览新闻。在该页左边放置一个新闻类别导航栏，单击某个新闻类别可在页面右边显示相应新闻，包括新闻的标题、作者、发表日期、浏览次数及内容的前 65 个字符。若要查看详细内容，需要单击标题提供的链接转到新闻详细页。在默认情况下，新闻速览页右边列表显示出所有新闻，并按 Id 降序排列，即最新的新闻显示在最前面。当然，也可根据需要按新闻的标题、日期分别进行升/降序排列。

在 Web 项目的 Admin 文件夹下，新建页面文件 NewsList.aspx。新闻速览页主要通过以下两步来完成。

1. 设置新闻类别导航栏和排序命令按钮

基本步骤如下：

(1) 添加新闻类别导航栏。向页面拖放一个 XmlDataSource 控件和一个 TreeView 控件并配置相应的属性。XmlDataSource 用于从 XML 文件 NewsCategoryTreeView.xml 中获取新闻类别数据，同时也作为 TreeView 控件的数据源。

(2) 添加排序命令按钮。向页面拖放两个 Button 控件，分别设置其 ID 属性值为 btnTitleSort 和 btnPubDateSort，分别用于按标题排序和按日期排序显示新闻。

2. 使用 DataList 控件显示新闻速览列表

基本步骤如下：

(1) 添加 DataList 控件。将工具箱“数据”选项卡中的 DataList 控件拖放到页面中，设置 DataList 控件的 ID 属性值为 dlNews。单击 DataList 控件右上角的小三角按钮打开 DataList 任务菜单，选择“编辑模板”命令，在 DataList 提供的模板项里自由布局各字段的显示方式。比如在项模板 ItemTemplate 里加入一个 3 行 3 列的表格布局，并设置各列的绑定字段，在分隔符模板 SeparatorTemplate 里加入一条水平线用于分隔各列表项。设计视图效果如图 6-1 所示。



图 6-1　新闻速览页未分页前设计视图

(2) 编码指定数据源。为了能够给 DataList 列表实现分页功能，需要在 NewsList.aspx.cs 中手动编写代码动态绑定数据源，其中用到 DataList 控件的 DataSource 属性和 DataBind()方法。

新闻速览页未分页前 NewsList.aspx 的关键代码如下所示：

```
<%@ Page Language = "C#" MasterPageFile = "~/NewsPage.Master" AutoEventWireup = "true"
CodeBehind = "NewsList.aspx.cs" Inherits = "Web.NewsList" Title = "新闻速览" %>
<asp:Content ID = "Content1" ContentPlaceHolderID = "ContentPlaceHolder1" runat = "server">
    <div id = "col1_list">
```

```
            <div class = "box">
                <div class = "box_title">新闻导航</div>
                <div class = "box_text">
                     <asp:TreeView ID = "tvNewsCategories" runat = "server" DataSourceID =
"XmlDataSource1" EnableViewState = "false">
                        <DataBindings>
                            <asp:TreeNodeBinding DataMember = "siteMapNode" NavigateUrlField =
"url" TextField = "title" />
                        </DataBindings>
                    </asp:TreeView>
                    <asp:XmlDataSource ID = "XmlDataSource1" runat = "server" DataFile = "~/
NewsCategoryTreeView.xml"></asp:XmlDataSource>
                </div>
            </div>
        </div>
        <div id = "col2_list">
            <div class = "box">
                <div class = "box_title"><asp:Label ID = "lblNewsCategory" runat = "server"
Text = "新闻速览" EnableViewState = "false"></div>
                <div class = "box_text">
                    <div>排序方式: <asp:Button ID = "btnTitleSort" runat = "server" Text =
"按标题升序" OnClick = "btnTitleSort_Click" />|<asp:Button ID = "btnPubDateSort" runat =
"server" Text = "按日期升序" OnClick = "btnPubDateSort_Click" />
                    </div>
                    <asp:DataList ID = "dlNews" runat = "server" EnableViewState = "false">
                        <ItemTemplate>
                            <table>
                                <tr><td colspan = "3">
                                        <span id = "title_span">
                                            <a href = "NewsDetail.aspx? Id = <% # Eval
("Id") %> "><% # Web.Common.StringHandler.FilterString(Eval("Title").ToString(), 65)
%></a></span></td></tr>
                                <tr><td>作者: <% # Eval("Author") %></td>
                                    <td>发表日期: <% # Eval("PubDate") %></td>
                                    <td>浏览次数: <% # Eval("Clicks") %></td></tr>
                                  <tr><td colspan = "3"><% # Web.Common.StringHandler.
FilterString(Eval("Contents").ToString(), 65) %></td></tr>
                            </table>
                        </ItemTemplate>
                        <SeparatorTemplate><hr /></SeparatorTemplate>
                    </asp:DataList>
                </div>
            </div>
        </div>
</asp:Content>
```

这里提醒一点，为了页面美观，经常用到将列表中绑定的某个字符串字段截断或过滤其中的 HTML 标记的技术。在上面的代码中，Web.Common.StringHandler.FilterString(Eval("Contents").ToString()，65)就是用于将新闻内容字段进行截取和过滤。FilterString()

是放在 Web\Common 文件夹下自定义类 StringHandler.cs 里的一个自定义方法。具体实现代码如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Text.RegularExpressions;
namespace Web.Common
{
    public static class StringHandler
    {
        /// <summary>
        /// 截取指定长度的子字符串
        /// </summary>
        /// <param name = "str">原字符串</param>
        /// <param name = "length">子字符串的长度</param>
        /// <returns>子字符串</returns>
        public static string CutString(string str, int length)
        {
            if (str.Length >= length)
                return str.Substring(0, length) + "...";
            else
                return str;
        }
        /// <summary>
        /// 用正则表达式过滤 HTML 字符
        /// </summary>
        /// <param name = "str">需要处理的字符串</param>
        /// <returns></returns>
        public static string RemoveHtml(string str)
        {
            return Regex.Replace(str, "</?[^>]*>|\\s", "", RegexOptions.Compiled |
RegexOptions.IgnoreCase);
        }
        /// <summary>
        /// 截取前先过滤 HTML 字符
        /// </summary>
        /// <param name = "str">原字符串</param>
        /// <param name = "length">子字符串的长度</param>
        /// <returns>子字符串</returns>
        public static string FilterString(string str, int length)
```

```
            {
                return CutString(RemoveHtml(str), length);
            }
        }
    }
```

NewsList.aspx.cs 的代码如下所示：

```
...
using NewsBLL;
namespace Web
{
    public partial class NewsList : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                //首次加载,赋初值
                if (Request.QueryString["NewsCategoryId"] != null)
                {
                    int id;
                    try
                    {
                        id = Convert.ToInt32(Request.QueryString["NewsCategoryId"]);
                        ViewState["Conditions"] = "NewsCategoryId=" + id;
                        lblNewsCategory.Text =
NewsCategoryManager.GetNewsCategoryById(id).Name;
                    }
                    catch
                    {
                        ViewState["Conditions"] = "";
                    }
                }
                else
                {
                    ViewState["Conditions"] = "";
                }
                ViewState["SortField"] = "Id";
                ViewState["Direction"] = "DESC";
                ViewState["TitleClick"] = "0";
                ViewState["PubDateClick"] = "0";
                Databind();
            }
        }
        private void Databind()
        {
            dlNews.DataSource =
NewsManager.GetNewsAllFieldsByConditions(ViewState["Conditions"].ToString(), ViewState
```

```
["SortField"].ToString(), ViewState["Direction"].ToString());
            dlNews.DataBind();
        }
        #region  排序
        protected void btnTitleSort_Click(object sender, EventArgs e)
        {
            ViewState["SortField"] = "Title";
            if (ViewState["TitleClick"].ToString() == "0")
            {
                ViewState["TitleClick"] = "1";
                ViewState["Direction"] = "ASC";
                btnTitleSort.Text = "按标题降序";
            }
            else if (ViewState["TitleClick"].ToString() == "1")
            {
                ViewState["TitleClick"] = "0";
                ViewState["Direction"] = "DESC";
                btnTitleSort.Text = "按标题升序";
            }
            Databind();
        }
        protected void btnPubDateSort_Click(object sender, EventArgs e)
        {
            ViewState["SortField"] = "PubDate";
            if (ViewState["PubDateClick"].ToString() == "0")
            {
                ViewState["PubDateClick"] = "1";
                ViewState["Direction"] = "ASC";
                btnPubDateSort.Text = "按日期降序";
            }
            else if (ViewState["PubDateClick"].ToString() == "1")
            {
                ViewState["PubDateClick"] = "0";
                ViewState["Direction"] = "DESC";
                btnPubDateSort.Text = "按日期升序";
            }
            Databind();
        }
        #endregion
    }
}
```

新闻速览页未分页前运行效果如图 6-2 所示。

6.1.4　小结

(1) DataList 控件以模板和样式定义的格式显示数据。

(2) Regex 类属于 System.Text.RegularExpressions 命名空间。Regex.Replace() 方法表示在指定的输入字符串内使用指定的替换字符串替换与指定正则表达式匹配的所

新闻速览
排序方式： 按标题升序 | 按日期升序
第四届"职场之星"模拟求职大赛
作者：青木 发表日期：2009/8/19 14:51:20 浏览次数：4
本次比赛共分三个环节，自我介绍、个人面试和团队面试。在第一轮的自我介绍中，选手们自信亮相、准备充分，端庄的仪态、流利的语言表达凸显...
100多家单位聚首温职院"招贤纳士"
作者：乔仁慧 发表日期：2009/8/19 14:40:52 浏览次数：2
8月18日，温州职业技术学院校园里一日三场招聘会让2010届毕业生们着实开心了一把，而制造类企业争先恐后来前来设摊，提供岗位招聘应...
以旧换新拉动新车消费
作者：吴晓 发表日期：2009/7/2 15:00:00 浏览次数：0
全国现有黄标车1800万辆待换。汽车以旧换新不仅是拉动消费的动力，也是为了环境保护、节约资源、节约能源、带动就业等多方面的目的，所...
留学要以"兴趣为先"
作者：张凡 发表日期：2009/7/2 14:49:00 浏览次数：0
无论在国内还是国外上大学，选择专业都是相当重要，这个选择可以说是职业生涯规划的起点，所以专业选择更应该慎重再慎重，尤其要考虑个人将...
公告牌Hot100单曲榜50周年
作者：叶芸 发表日期：2009/7/2 11:01:00 浏览次数：0
1958年的8月4日，公告牌杂志设立了一个具有革命性的单曲排行榜--Hot100单曲榜。在经过近20年的发展之后，Hot100单曲...
《冰河世纪3》香港首映
作者：tungstar 发表日期：2009/7/2 10:40:00 浏览次数：0
北京时间7月1日消息，6月30日，香港圆方戏院举行《冰河世纪3》慈善首映礼，香港艺人官恩娜、陆永、林晓峰、赌王之女何超云，以及数位...
中国女队跻身接力决赛
作者：Belldandy 发表日期：2009/7/2 10:39:00 浏览次数：0
在北京时间今天下午进行的游泳世锦赛4乘200米自由泳接力赛预赛中，中国队以第六名的成绩进入决赛。英国队发挥出色，以打破欧洲纪录和赛...
盘点新世纪网坛10冷门
作者：王卓 发表日期：2009/7/2 10:38:00 浏览次数：0
结果的不可预知性一直是竞技体育的一大魅力所在，网坛亦是如此，接下来将为您盘点网坛自2000年以来新世纪的十大冷门，在体育圈&ldq...
多数上市银行已超额完成全年信贷目标
作者：钟正 发表日期：2009/7/2 10:37:00 浏览次数：0
截至5月末，有可比数据的9家主要商业银行累计新增贷款投放量为3.65万亿元，为全年计划信贷投放目标2.76万亿元的1.3倍，表明上...

图 6-2　新闻速览页未分页前运行效果

有字符串。可用它来清除 HTML 标记。

6.1.5　思考与练习

1. 如何通过程序控制动态地为 DataList 控件绑定数据源？
2. 要将某个字段内容中的 HTML 标记清除后，再显示页面上，应该如何操作？

任务 6.2　使用 PagedDataSource 分页显示新闻

任务目标

会编写代码实现分页。

6.2.1　PagedDataSource 对象简介

DataList 控件没有提供内置分页功能，有时很不方便。目前有很多增加分页的方法，比如使用存储过程来控制每页的数据读取，这种分页方法效率高但制作起来比较麻烦，下面介绍一种可以快速实现分页的方法：使用 PagedDataSource 对象给 DataList 增加分页功能。PagedDataSource 类封装了数据绑定控件的与分页相关的属性，以允许该控件执行分页操作。PagedDataSource 类的常用属性如表 6-2 所示。

表 6-2　PagedDataSource 类的常用属性

属　　性	说　　明
AllowPaging	获取或设置一个值，指示是否在数据绑定控件中启用分页
Count	获取要从数据源使用的项数
DataSourceCount	获取数据源中的项数
CurrentPageIndex	获取或设置当前页的索引
DataSource	获取或设置数据源
IsFirstPage	获取一个值，该值指示当前页是否是首页
IsLastPage	获取一个值，该值指示当前页是否是最后一页
PageCount	获取显示数据源中的所有项所需要的总页数
PageSize	获取或设置要在单页上显示的项数

6.2.2　使用 PagedDataSource 实现新闻速览页分页显示

打开任务 6.1 完成的新闻速览页 NewsList.aspx，在该页 DataList 控件下方增加与分页相关的标签和导航按钮，设计视图如图 6-3 所示。



图 6-3　为新闻速览页增加分页导航按钮后的设计视图

以下是对应的 HTML 代码：

```
<div id="page">
    <asp:Label ID="lblPage" runat="server" Text=""></asp:Label>
    <asp:Button ID="btnFirst" runat="server" Text="首页" OnClick="btnFirst_Click" />
    <asp:Button ID="btnPrev" runat="server" Text="上一页" OnClick="btnPrev_Click" />
    <asp:Button ID="btnNext" runat="server" Text="下一页" OnClick="btnNext_Click" />
    <asp:Button ID="btnLast" runat="server" Text="末页" OnClick="btnLast_Click" />
</div>
```

在 NewsList.aspx.cs 中编写后台代码实现分页功能。

(1) 修改 DataBind()方法

重新设定 DataList 控件 pdsNews 的数据源为 PagedDataSource 对象。实现代码如下：

```
private void Databind()
{
    //设置 PagedDataSource 对象实例
    PagedDataSource pdsNews = new PagedDataSource();
    //设置数据源
    pdsNews.DataSource = NewsManager.GetNewsAllFieldsByConditions(ViewState["Conditions"].
ToString(), ViewState["SortField"].ToString(), ViewState["Direction"].ToString());
```

```
        //设置允许分页
        pdsNews.AllowPaging = true;
        //设置每页显示条记录
        pdsNews.PageSize = 5;
        //设置当前页索引值
        pdsNews.CurrentPageIndex = PageIndex;
        //设置末页索引值
        ViewState["LastPageIndex"] = pdsNews.PageCount - 1;
        //设置分页导航栏控件状态
        SetControlState(pdsNews);
        //绑定数据源
        dlNews.DataSource = pdsNews;
        dlNews.DataBind();
    }
```

(2) 设置分页导航按钮事件和状态

为方便引用,将当前页索引值 ViewState["PageIndex"]设置成属性 PageIndex。实现代码如下:

```
#region 翻页
/// <summary>
/// 当前页索引值(设成属性便于在程序访问)
/// </summary>
private int PageIndex
{
    get
    {
        return (int)ViewState["PageIndex"];
    }
    set
    {
        ViewState["PageIndex"] = value;
    }
}
protected void btnFirst_Click(object sender, EventArgs e)
{
    PageIndex = 0;
    Databind();
}
protected void btnPrev_Click(object sender, EventArgs e)
{
    PageIndex--;
    Databind();
}
protected void btnNext_Click(object sender, EventArgs e)
{
    PageIndex++;
    Databind();
}
```

```
protected void btnLast_Click(object sender, EventArgs e)
{
    PageIndex = Convert.ToInt32(ViewState["LastPageIndex"]);
    Databind();
}
private void SetControlState(PagedDataSource pds)
{
    int recNum = pds.DataSourceCount;
    if (recNum > 0) //数据源中有记录
    {
        btnTitleSort.Enabled = true;
        btnPubDateSort.Enabled = true;
        btnFirst.Enabled = true;
        btnPrev.Enabled = true;
        btnNext.Enabled = true;
        btnLast.Enabled = true;
        if (pds.IsFirstPage)    //当前页为首页时,设置"首页"、"上一页"按钮无效
        {
            btnFirst.Enabled = false;
            btnPrev.Enabled = false;
        }
        if (pds.IsLastPage)     //当前页为末页时,设置"末页"、"下一页"按钮无效
        {
            btnNext.Enabled = false;
            btnLast.Enabled = false;
        }
        //标签控件中显示当前页及总页数等信息
        lblPage.Text = "第" + (pds.CurrentPageIndex + 1) + " 页共" + pds.PageCount +
" 页共" + recNum + " 条记录";
    }
    else
    {
        btnTitleSort.Enabled = false;
        btnPubDateSort.Enabled = false;
        btnFirst.Enabled = false;
        btnPrev.Enabled = false;
        btnNext.Enabled = false;
        btnLast.Enabled = false;
        lblPage.Text = "记录未找到";
    }
}
#endregion
```

(3) 给当前页索引赋初值

当页面首次加载时需要给当前页索引赋初值。所以在 Page_Load 事件的 if (!IsPostBack){…}代码里设置“ViewState["PageIndex"] = 0;”。

在排序操作后应将当前页码重置到第一页,所以在 btnTitleSort_Click 和 btnPubDateSort_Click 事件中分别设置“PageIndex = 0;”。

分页后的新闻速览页最终运行效果如图 6-4 所示。

图 6-4　新闻速览页最终运行效果

6.2.3　小结

（1）使用 PagedDataSource 对象实现数据分页显示，做法相对来说比较简单。但它有一个很大的缺陷，即每换一次页，必须重新访问数据库，再次将表中所有数据加载到缓存中，如果数据量很大，有几万行的话，这个缺陷就成致命的了。所以这种方法可用于数据量不大的情况下。

（2）使用 PagedDataSource 对象的基本步骤为：创建一个 PagedDataSource 类的对象实例→给 PagedDataSource 对象的相关属性赋值→将 PagedDataSource 对象绑定到数据绑定控件。

6.2.4　思考与练习

1. 使用 PagedDataSource 对象实现数据分页有哪几个步骤？

2. 基于存储过程的分页和基于 PagedDataSource 的分页各有什么优缺点？分别适用于什么场合？

任务 6.3　使用 Repeater 列表显示新闻

任务目标

会用 Repeater 控件自定义模板显示数据。

6.3.1　Repeater 控件简介

Repeater 控件是一个基本模板数据绑定列表控件。它与 DataList 控件相似,可用来显示被绑定数据项的一个循环序列,都没有内置分页功能。两者也有不同:DataList 控件提供了布局功能,会自动在数据项周围生成一个 HTML 表格,有内置的选择和编辑功能,而 Repeater 控件则不然。Repeater 控件功能比 DataList 控件简单,它没有内置的布局或样式,不会自动生成任何 HTML 标签,因此必须在这个控件的模板内显式声明所有的 HTML 布局标记、格式设置及样式标记等。Repeater 控件没有内置的选择和编辑功能,它只能用于数据显示。也正因为此,Repeater 控件具有更好的灵活性和更高的效率。不过,Repeater 控件的所有代码必须在 Web 页面的源代码视图中手工添加。

Repeater 控件的数据显示形式完全由用户通过模板来控制。例如,若通过 HTML 表格显示数据,应在 HeaderTemplate 模板中放置<table>标记来开始此表格,然后在 ItemTemplate 模板中放置<tr>和<td>及数据绑定项来创建该表的行和列。若要使表格中交替项呈现不同的外观,可使用与 ItemTemplate 相同的内容创建 AlternatingItem-Template 模板,并为其指定一个不同的样式。最后在 FooterTemplate 模板中放置</table>标记完成表格的设置。Repeater 控件支持的模板如表 6-3 所示。

表 6-3　Repeater 控件支持的模板

模板属性	说　　明
ItemTemplate	包含要为数据源中每个数据项都要呈现一次的 HTML 元素和控件
AlternatingItemTemplate	包含要为数据源中每个数据项都要呈现一次的 HTML 元素和控件。通常,可以使用此模板为交替项创建不同的外观,例如指定一种与在 ItemTemplate 中指定的颜色不同的背景色
HeaderTemplate	包含在列表的开始处呈现的文本和控件,如标题、列标头等
FooterTemplate	包含在列表的结束处呈现的文本和控件,如脚注的内容和布局
SeparatorTemplate	包含在每项之间呈现的元素,如使用<hr />标记作一条分隔线

6.3.2　使用 Repeater 控件实现新闻搜索列表显示

在新闻搜索页面里,可按新闻标题进行搜索,搜索结果用 Repeater 控件以列表方式显示。为查看方便起见,列表里显示除新闻内容(Contents)外的部分字段,要求各交替项应用不同的背景色,也能排序和分页。使用 Repeater 控件列表显示数据,做法上与 DataList 控件类似,新闻搜索页的做法类似于新闻速览页,也有排序、分页功能,在此不再赘述。下面只列出使用 Repeater 控件显示新闻搜索列表的基本步骤。

在 Web 项目的根目录下,新建页面文件 NewsSearch.aspx。

使用 Repeater 控件显示新闻搜索列表的基本步骤如下。

(1) 添加 Repeater 控件。将工具箱“数据”选项卡中的 Repeater 控件拖放到页面中,设置 Repeater 控件的 ID 属性值为 rpNews。

(2) 在“源代码”视图中编写模板代码。切换到页面的“源代码”视图,在<asp:Repeater>和</asp:Repeater>标记之间编写 ItemTemplate、AlternatingItemTemplate

模板代码。这里采用<ul>列表标记结合样式控制来定义 Repeater 列表布局。

（3）编写后台代码绑定数据源。

NewsSearch.aspx 实现代码如下：

```
<%@ Page Language="C#" MasterPageFile="~/NewsPage.Master" AutoEventWireup="true"
CodeBehind="NewsSearch.aspx.cs" Inherits="Web.NewsSearch" Title="新闻搜索" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div id="col">
        <div class="box">
            <div class="box_title">新闻搜索</div>
            <div class="box_text">
                <div>排序方式：<asp:Button ID="btnTitleSort" runat="server" Text=
"按标题升序" OnClick="btnTitleSort_Click" />|<asp:Button ID="btnPubDateSort" runat=
"server" Text="按日期升序" OnClick="btnPubDateSort_Click" />
                </div>
                <div>
                    <ul id="title_ul">
                        <li class="li0">标题</li>
                        <li class="li1">作者</li>
                        <li class="li2">日期</li>
                        <li class="li3">浏览次数</li>
                        <li>类别</li>
                    </ul>
                    <asp:Repeater ID="rpNews" runat="server" EnableViewState="false">
                    <ItemTemplate>
                        <ul id="content_ul1">
                            <li class="li0"><a href="NewsDetail.aspx?Id=<%# Eval
("Id")%>"><%# Web.Common.StringHandler.FilterString(Eval("Title").ToString(), 25) %>
</a></li>
                            <li class="li1"><%# Eval("Author") %></li>
                            <li class="li2"><%# Eval("PubDate") %></li>
                            <li class="li3"><%# Eval("Clicks") %></li>
                            <li><%# Eval("NewsCategory.Name")%></li>
                        </ul>
                    </ItemTemplate>
                    <AlternatingItemTemplate>
                        <ul id="content_ul2">
                            <li class="li0"><a href="NewsDetail.aspx?Id=<%# Eval
("Id")%>"><%# Web.Common.StringHandler.FilterString(Eval("Title").ToString(), 25) %>
</a></li>
                            <li class="li1"><%# Eval("Author") %></li>
                            <li class="li2"><%# Eval("PubDate") %></li>
                            <li class="li3"><%# Eval("Clicks") %></li>
                            <li><%# Eval("NewsCategory.Name")%></li>
                        </ul>
                    </AlternatingItemTemplate>
                    </asp:Repeater>
                </div>
                <div id="page">
```

```
                    < asp:Label ID = "lblPage" runat = "server" Text = "" EnableViewState =
"false"></asp:Label >
                    < asp:Button ID = "btnFirst" runat = "server" Text = "首页" OnClick =
"btnFirst_Click" />
                    < asp:Button ID = "btnPrev" runat = "server" Text = "上一页" OnClick =
"btnPrev_Click" />
                    < asp:Button ID = "btnNext" runat = "server" Text = "下一页" OnClick =
"btnNext_Click" />
                    < asp:Button ID = "btnLast" runat = "server" Text = "末页" OnClick =
"btnLast_Click" />
                </div >
            </div >
        </div >
    </div >
</asp:Content >
```

NewsSearch.aspx.cs 实现代码如下：

```
…
using NewsBLL;
namespace Web
{
    public partial class NewsSearch : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                //首次加载,赋初值
                if (Request.QueryString["Title"] != null)
                {
                    string title = Server.UrlDecode(Request.QueryString["Title"]);
                    ViewState["Title"] = "Title LIKE '%" + title + "%'";
                    TextBox txtTitleSearch = (TextBox)Master.FindControl("txtTitleSearch");
                    if (txtTitleSearch != null)
                    {
                        txtTitleSearch.Text = title;
                    }
                }
                else
                {
                    ViewState["Title"] = "";
                }
                ViewState["PageIndex"] = 0;
                ViewState["SortField"] = "Id";
                ViewState["Direction"] = "DESC";
                ViewState["TitleClick"] = "0";
                ViewState["PubDateClick"] = "0";
                Databind();
            }
```

```
}
private void Databind()
{
    PagedDataSource pdsNews = new PagedDataSource();
    //给 PagedDataSource 对象的相关属性赋值
        pdsNews.DataSource = NewsManager.GetNewsPartFieldsByConditions(ViewState
["Title"].ToString(), ViewState["SortField"].ToString(), ViewState["Direction"].ToString());
    pdsNews.AllowPaging = true;
    pdsNews.PageSize = 5;
    pdsNews.CurrentPageIndex = PageIndex;
    ViewState["LastPageIndex"] = pdsNews.PageCount - 1;
    SetControlState(pdsNews);
    //把 PagedDataSource 对象赋给 DataList 控件
    rpNews.DataSource = pdsNews;
    rpNews.DataBind();
}
#region  排序
protected void btnTitleSort_Click(object sender, EventArgs e)
{
    ViewState["SortField"] = "Title";
    PageIndex = 0;
    if (ViewState["TitleClick"].ToString() == "0")
    {
        ViewState["TitleClick"] = "1";
        ViewState["Direction"] = "ASC";
        btnTitleSort.Text = "按标题降序";
    }
    else if (ViewState["TitleClick"].ToString() == "1")
    {
        ViewState["TitleClick"] = "0";
        ViewState["Direction"] = "DESC";
        btnTitleSort.Text = "按标题升序";
    }
    Databind();
}
protected void btnPubDateSort_Click(object sender, EventArgs e)
{
    ViewState["SortField"] = "PubDate";
    PageIndex = 0;
    if (ViewState["PubDateClick"].ToString() == "0")
    {
        ViewState["PubDateClick"] = "1";
        ViewState["Direction"] = "ASC";
        btnPubDateSort.Text = "按日期降序";
    }
    else if (ViewState["PubDateClick"].ToString() == "1")
    {
        ViewState["PubDateClick"] = "0";
        ViewState["Direction"] = "DESC";
```

```
        btnPubDateSort.Text = "按日期升序";
    }
    Databind();
}#endregion

#region  翻页
/// <summary>
/// 当前页数
/// </summary>
private int PageIndex
{
    get
    {
        return (int)ViewState["PageIndex"];
    }
    set
    {
        ViewState["PageIndex"] = value;
    }
}
protected void btnFirst_Click(object sender, EventArgs e)
{
    PageIndex = 0;
    Databind();
}
protected void btnPrev_Click(object sender, EventArgs e)
{
    PageIndex--;
    Databind();
}
protected void btnNext_Click(object sender, EventArgs e)
{
    PageIndex++;
    Databind();
}
protected void btnLast_Click(object sender, EventArgs e)
{
    PageIndex = Convert.ToInt32(ViewState["LastPageIndex"]);
    Databind();
}
private void SetControlState(PagedDataSource pds)
{
    int recNum = pds.DataSourceCount;
    if (recNum > 0)
    {
        btnTitleSort.Enabled = true;
        btnPubDateSort.Enabled = true;
        btnFirst.Enabled = true;
        btnPrev.Enabled = true;
```

```
                    btnNext.Enabled = true;
                    btnLast.Enabled = true;
                    if (pds.IsFirstPage)
                    {
                        btnFirst.Enabled = false;
                        btnPrev.Enabled = false;
                    }
                    if (pds.IsLastPage)
                    {
                        btnNext.Enabled = false;
                        btnLast.Enabled = false;
                    }
                    lblPage.Text = "第" + (pds.CurrentPageIndex + 1) + " 页共" + pds.
PageCount + " 页共" + recNum + " 条记录";
                }
                else
                {
                    btnTitleSort.Enabled = false;
                    btnPubDateSort.Enabled = false;
                    btnFirst.Enabled = false;
                    btnPrev.Enabled = false;
                    btnNext.Enabled = false;
                    btnLast.Enabled = false;
                    lblPage.Text = "记录未找到";
                }
            } #endregion
        }
    }
```

注意：由于在页面设计时将搜索框 txtTitleSearch 置于母版页中，所以在内容页 NewsSearch.aspx 中访问母版页的控件时，采用 Master.FindControl()方法获取。新闻搜索页最终运行效果如图 6-5 所示。

6.3.3 小结

(1) Repeater 控件使用数据源返回的一组记录呈现只读列表。Repeater 控件不具备内置的呈现功能，用户必须通过创建模板来自定义数据显示布局。当页面运行时，该控件为数据源中的每个项重复此布局。

(2) Repeater 控件没有默认的外观，不会自动生成任何 HTML 标签，无内置分页功能。只能手工编写其模板代码和分页功能。

(3) ASP.NET 的 5 大数据绑定控件 GridView、DetailsView、FormView、DataList、Repeater 的比较：GridView、DetailsView、FormView 控件都是 ASP.NET 2.0 新增的控件，内置了分页等功能；GridView、DataList、Repeater 控件用于呈现多条记录，DetailsView、FormView 控件用于呈现单条记录明细；GridView、DetailsView 控件的布局固定，自定义数据显示的布局功能有限，一般适合布局简单的数据呈现。FormView、DataList、Repeater 控件都有很强的自定义布局能力，如果数据呈现需要较为复杂的布局



图 6-5　新闻搜索页最终运行效果

方案，这 3 个控件是首选。

6.3.4　思考与练习

1. 用 Repeater 控件完成首页中 15 条最新新闻和 15 条最新留言列表显示。
2. 用 Repeater 控件完成用户留言页中留言列表显示。

第 7 章　XML 访问技术

任务 7.1　一个基于 XML 的留言板设计

任务目标

（1）了解 ASP.NET 访问 XML 数据所使用的常用处理类。

（2）掌握使用 XmlDataSource 数据源控件和数据绑定控件访问 XML 数据的基本方法。

7.1.1　访问 XML 的常用处理类

XML(eXtensible Markup Language,可扩展的标记语言)类似于 HTML,它也是一种用来描述数据的标记语言。XML 的优势在于它可以预定义数据结构、扩展标记集;可以跨平台使用,并且可以以数据源的形式存在于服务器端。

.NET Framework 提供了丰富的组件和类来实现对 XML 数据的访问。在.NET 类库中,实现对 XML 文档进行操作的最常用的扩展类是 XmlDocument 类,它是由基础类 XmlNode 派生的。

XmlDocument 类的常用属性、方法及相关说明如表 7-1、表 7-2 所示。

表 7-1　XmlDocument 类的常用属性

属　　性	说　　明
DocumentElement	文档的根
ChildNodes	节点的所有子节点
ParentNode	节点的父级节点
LastChild	节点的最后一个子级节点
InnerXml	节点所包含的所有 XML 内容
Name	节点名称

表 7-2　XmlDocument 类的常用方法

方　　法	说　　明
Load	加载 XML 文档
CreateElement	创建一个指定的元素
CreateTextNode	创建具有指定文本的 XmlText
CreateNode	创建一个指定的 XmlNode
AppendChild	将指定节点添加到该节点的子节点列表的末尾
SelectNodes	选择匹配的节点
RemoveChild	移除指定的子节点
Save	保存 XML 文档到指定位置

下面通过一个简单的基于 XML 技术的留言板来说明在 ASP.NET 中访问 XML 文档的方法和过程。

7.1.2　创建留言板的 XML 文件和 XSLT 文件

(1) 运行 Visual Studio 2005。在“解决方案资源管理器”面板中，右击“解决方案‘0931’”，在弹出的快捷菜单中，选择“添加”→“新建网站”命令，在打开的对话框中新建网站 Chap7。

右击站点名 Chap7，在弹出的快捷菜单中选择“添加新项”命令。在打开的“添加新项”对话框中选择“XML 文件”选项，可以使用默认文件名 XMLFile. xml。在 XMLFile. xml 文件中添加一个根级别的元素<message>。

XMLFile. xml 初始代码如下：

```
<?xml version = "1.0" encoding = "utf - 8" ?>
<message>
</message>
```

(2) 右击站点名 Chap7，在弹出的快捷菜单中选择“添加新项”命令。在打开的“添加新项”对话框中选择“XSLT 文件”选项，可以使用默认文件名 XSLTFile. xsl。

XSLT(可扩展样式语言)是 XML 文档的样式文件，它负责将 XML 文档转换成不同应用程序所能接受的数据。

在 XSLTFile. xsl 文件中编写代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<xsl:stylesheet version = "1.0"
    xmlns:xsl = "http://www.w3.org/1999/XSL/Transform">
  <xsl:template match = "message">
    <xsl:element name = "message">
      <xsl:for - each select = "..//record">
        <xsl:element name = "record">
          <xsl:attribute name = "id">
             <xsl:value - of select = "id"/>
          </xsl:attribute>
          <xsl:attribute name = "name">
             <xsl:value - of select = "name"/>
          </xsl:attribute>
          <xsl:attribute name = "content">
             <xsl:value - of select = "content"/>
          </xsl:attribute>
          <xsl:attribute name = "msgtime">
             <xsl:value - of select = "msgtime"/>
          </xsl:attribute>
        </xsl:element>
      </xsl:for - each>
    </xsl:element>
  </xsl:template>
</xsl:stylesheet>
```

7.1.3 XML 访问的公共类设计

运行 Visual Studio 2005。在“解决方案资源管理器”面板中，右击站点 Chap7 下的 App_Code 文件夹，在弹出的快捷菜单中选择“添加新项”命令。在打开的“添加新项”对话框中选择“类”选项，文件名存为 XmlWrite. cs。

XmlWrite. cs 中主要实现了添加留言、删除留言的功能。主要实现代码如下：

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Xml;

/// <summary>
/// XmlWrite 的摘要说明
/// </summary>

public class XmlWrite
{
    public XmlWrite()
    {
        // TODO: 在此处添加构造函数逻辑
    }

    /// <summary>
    /// 写 XML 文件
    /// </summary>
    public void WriteXML(string FileName, string id, string name, string content, string
msgtime)
    {
        //创建 XmlDocument 对象实例
        XmlDocument doc = new XmlDocument();
        //加载 XML 文档
        doc.Load(FileName);

        //以下新建元素 - id,name,content,msgtime
        XmlElement ele_id = doc.CreateElement("id");
        XmlText text_id = doc.CreateTextNode(id);

        XmlElement ele_name = doc.CreateElement("name");
        XmlText text_name = doc.CreateTextNode(name);

        XmlElement ele_content = doc.CreateElement("content");
```

```
        XmlText text_content = doc.CreateTextNode(content);

        XmlElement ele_msgtime = doc.CreateElement("msgtime");
        XmlText text_msgtime = doc.CreateTextNode(msgtime);

        //以下新建子节点,并在节点中添加元素
        XmlNode newNode = doc.CreateNode("element", "record", "");

        newNode.AppendChild(ele_id);
        newNode.LastChild.AppendChild(text_id);

        newNode.AppendChild(ele_name);
        newNode.LastChild.AppendChild(text_name);

        newNode.AppendChild(ele_content);
        newNode.LastChild.AppendChild(text_content);

        newNode.AppendChild(ele_msgtime);
        newNode.LastChild.AppendChild(text_msgtime);

        //将子节点添加到 XML 文档中
        XmlElement root = doc.DocumentElement;
        root.AppendChild(newNode);

        //保存所有修改
        doc.Save(FileName);
    }

    /// <summary>
    /// 删除 XML 文件中的子节点
    /// </summary>
    public void DeleNote(string FileName, string PassNode)
    {
        XmlDocument doc = new XmlDocument();
        doc.Load(FileName);

        XmlElement root = doc.DocumentElement;//文档的根
        //寻找所有包含"id"元素的子节点
        XmlNodeList nodes = root.SelectNodes("//id");
        if (!(nodes == null))
        {
            //遍历所有包含"id"元素的子节点
            foreach (XmlNode node in nodes)
            {
                //找到包含当前"id" 值的节点元素
                if (node.InnerXml == PassNode.ToString())
                    //删除该节点元素的父节点
                    root.RemoveChild(node.ParentNode);
            }
```

```
            }
            //保存所有修改
            doc.Save(FileName);
        }
    }
```

7.1.4 使用 XmlDataSource 控件和 DataList 控件显示留言

运行 Visual Studio 2005。在“解决方案资源管理器”面板中，右击站点名 Chap7，在弹出的快捷菜单中选择“添加新项”命令。在打开的“添加新项”对话框中选择“Web 窗体”选项，文件名存为 Default.aspx。

切换到“设计”视图，为 Default.aspx 页面添加控件。从左侧工具箱数据组中拖出一个 XmlDataSource 控件和一个 DataList 控件。

右击 DataList 控件右上角的小三角按钮，在弹出的“DataList 任务”菜单中，选择“选择数据源”→“新建数据源”命令，打开“数据源配置向导”对话框，选择数据源类型为“XML 文件”，并在“为数据源指定 ID”文本框中输入 XmlDataSource1，如图 7-1 所示。

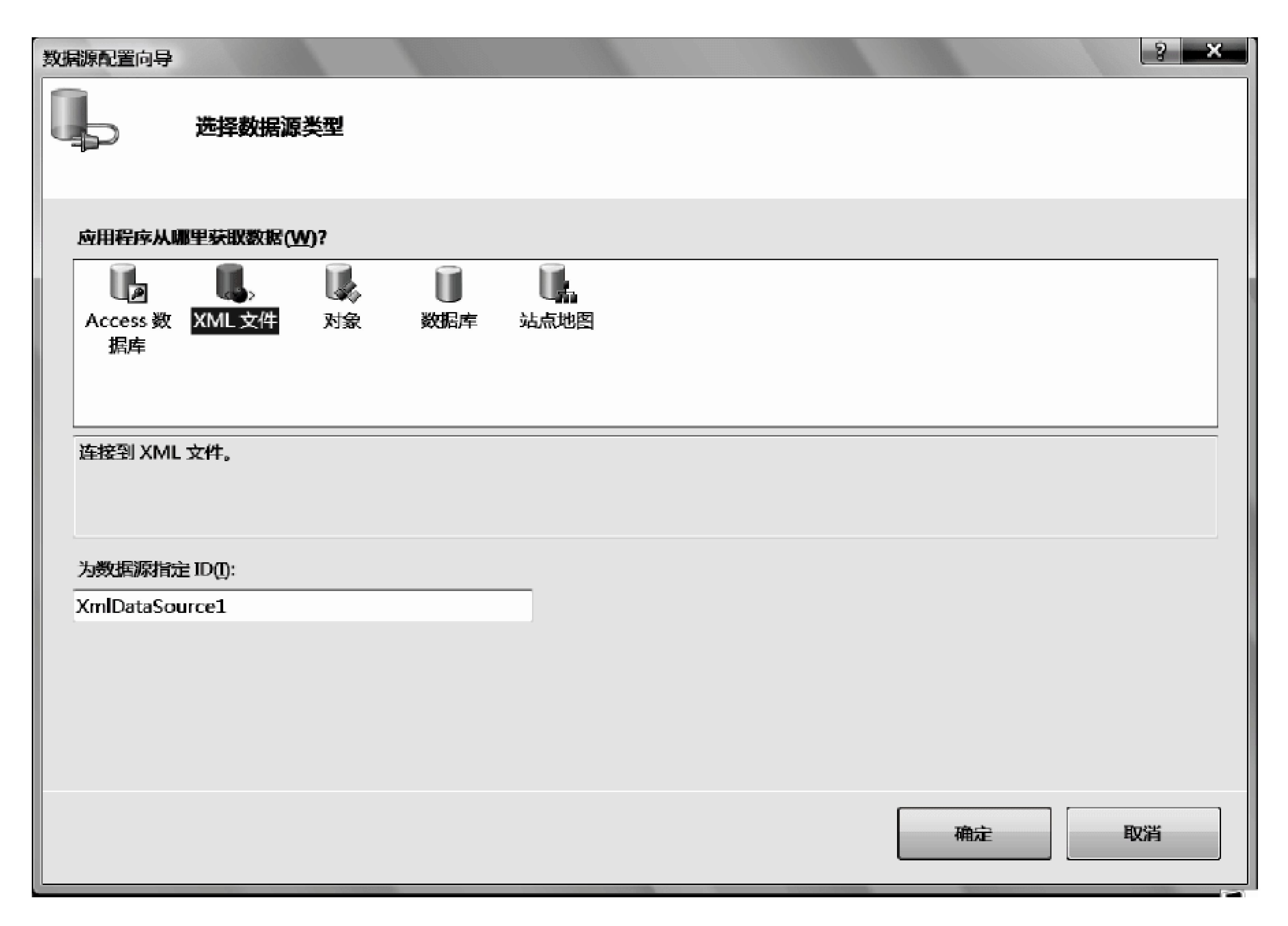


图 7-1 “数据源配置向导”对话框

单击“确定”按钮，在随后弹出的“配置数据源-XmlDataSource1”对话框中指定“数据文件”和数据“转换文件”，如图 7-2 所示。

Default.aspx 显示留言页面主要代码如下：

```
…
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"><title>0931 在线论坛</title></head><body>
```



图 7-2　配置数据源对话框

```
<form id = "form1" runat = "server">
    <div>
        <asp:DataList ID = "DataList1" runat = "server" DataSourceID = "XmlDataSource2"
BackColor = "#CCCCCC" BorderColor = "#999999" BorderStyle = "Solid" BorderWidth = "3px"
CellPadding = "4" CellSpacing = "2" ForeColor = "Black" GridLines = "Both" Width = "1050px">
            <ItemTemplate>
                序号:<asp:Label ID = "Label1" runat = "server" Text = '<%# Eval("id") %
>'></asp:Label><br />
                姓名: <asp:Label ID = "nameLabel" runat = "server" Text = '<%# Eval
("name") %>'></asp:Label><br />
                发言内容: <asp:Label ID = "msgLabel" runat = "server" Text = '<%# Eval
("content") %>'></asp:Label><br />
                留言时间: <asp:Label ID = "urlLabel" runat = "server" Text = '<%# Eval
("msgtime") %>'></asp:Label><br />
            </ItemTemplate>
        </asp:DataList>
        <asp:XmlDataSource ID = "XmlDataSource2" runat = "server" DataFile = "~/ XMLFile.
xml " TransformFile = "~/ XMLFile.xsl">
        </asp:XmlDataSource>
    </div>
</form>
</body></html>
```

Default.aspx 显示留言页面运行效果如图 7-3 所示。

7.1.5　添加留言到 XML 文件中

右击站点名 Chap7，在弹出的快捷菜单中选择“添加新项”命令。打开“添加新项”对

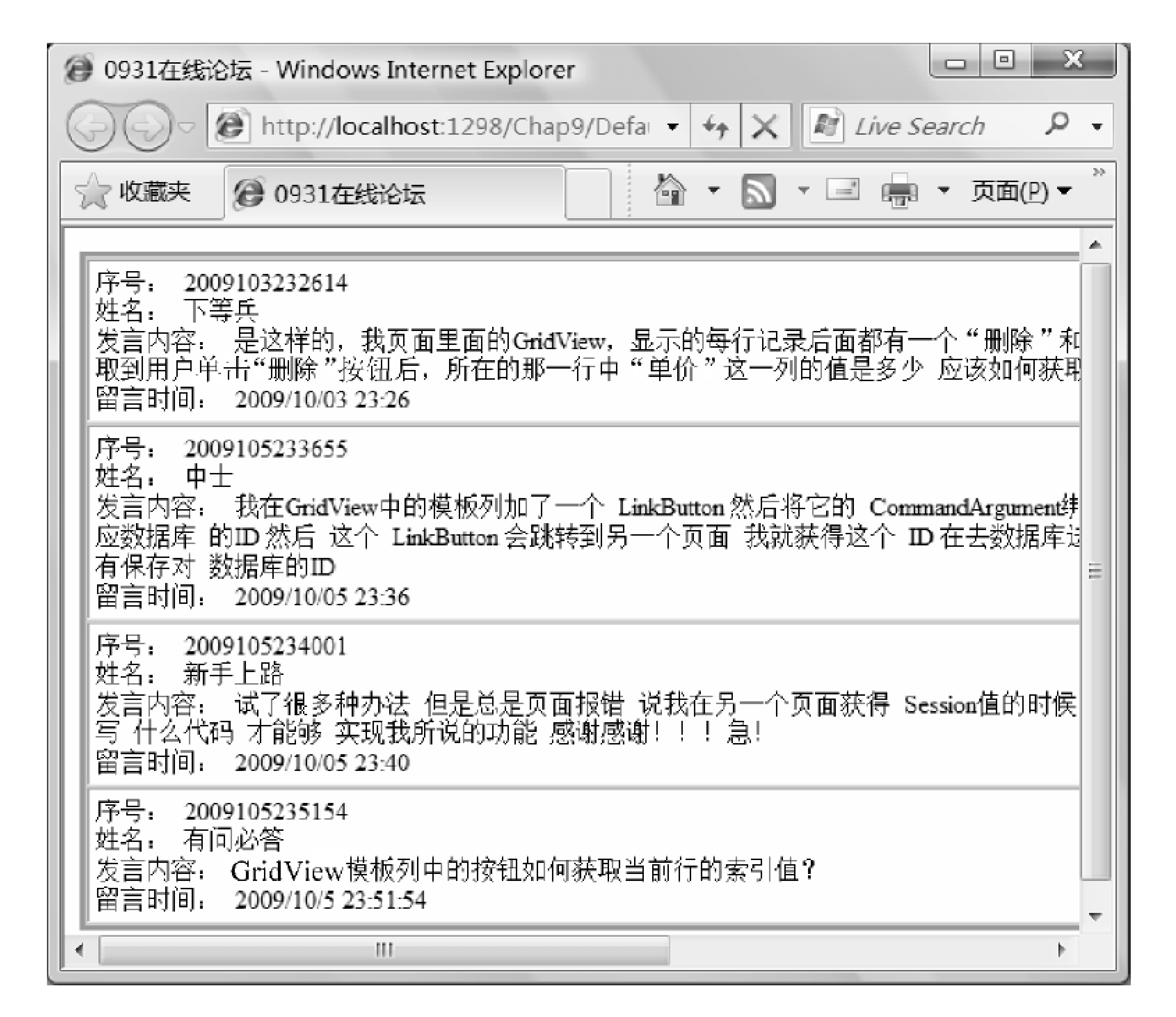


图 7-3 显示留言页面运行效果

话框，选择“Web 窗体”选项，添加留言页面文件名存为 AddMessage.aspx。

切换到“设计”视图，为 AddMessage.aspx 页面添加若干个 Label 控件、TextBox 控件、Button 控件以及 RequiredFieldValidator 必添控件。AddMessage.aspx 页面界面设计和运行效果如图 7-4 所示。



图 7-4 添加留言页面

在 AddMessage.aspx 页面中，使用了一个 Label2 控件，用来自动显示并记录留言发表的日期和时间，注意在 AddMessage.aspx.cs 文件的 Page_Load()事件中，需要用 Label2.Text =DateTime.Now.ToString();代码给 Label2 的 Text 属性赋值。

AddMessage.aspx 添加留言页面代码如下：

```
…
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server"><title>0931 在线论坛 - 添加留言</title></head>
<body>
<form id="form1" runat="server">
    <div>
        <table style="width: 416px">
            <tr><td>发言内容: </td></tr>
            <tr><td>
                <asp:TextBox ID = "TextBox2" runat = "server" Rows = "8" TextMode =
"MultiLine" Width="394px"></asp:TextBox>
                <asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
ControlToValidate="TextBox2" ErrorMessage="请填写留言内容"></asp:RequiredFieldValidator>
            </td></tr>
            <tr><td>姓名: <asp:TextBox ID="TextBox1" runat="server" Width="148px">
</asp:TextBox>
               <asp:RequiredFieldValidator ID="RequiredFieldValidator4" runat="server"
ControlToValidate="TextBox1" ErrorMessage="请填写姓名"></asp:RequiredFieldValidator>
            </td></tr>
            <tr><td style="height: 26px">
                <asp:Label ID="Label1" runat="server" Text="提交时间"></asp:Label>
                <asp:Label ID="Label2" runat="server" Text="显示时间"></asp:Label>
            </td></tr>
            <tr><td style="width: 230px">
                <asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="提
交留言" />
            </td></tr>
        </table>
    </div>
</form>
</body>
</html>
```

在 AddMessage.aspx.cs 中编写如下事件代码：

```
…
using System.Xml;
public partial class AddMessage : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Label2.Text = DateTime.Now.ToString();
    }
```

```
        protected void Button1_Click(object sender, EventArgs e)
        {
            if (TextBox1.Text ! = string.Empty &&TextBox2.Text! = string .Empty)
            {
                //获取发言时系统的日期和时间
                string date = DateTime .Now .Year.ToString() + DateTime .Now .Month.ToString() +
    DateTime .Now .Day.ToString();
                string time = DateTime .Now .Hour.ToString() + DateTime .Now .Minute.ToString() +
    DateTime .Now .Second.ToString();
                // 构建留言序号
                string MessgeID = date + time;
                //初始化 XmlWrite 类
                XmlWrite AddMessage = new XmlWrite();
                AddMessage.WriteXML(Server.MapPath("XMLFile.xml "), MessgeID.ToString(),
    TextBox1.Text, TextBox2.Text, Label2.Text);
                //重定向到首页浏览页面
                Response.Redirect("Default.aspx");
            }
        }
    }
```

说明：

① 注意对留言序号的处理，这里用发言时系统的日期和时间构建了留言 id。也可以用别的途径解决留言序号的问题。

② WriteXML()方法所需的第一个参数是 XMLFile.xml，后续 4 个参数依次为：留言序号(id)、姓名(name)、发言内容(content)、提交时间(msgtime)。

7.1.6 使用 XmlDataSource 控件和 GridView 控件删除留言

右击站点名 Chap7，在弹出的快捷菜单中选择“添加新项”命令，打开“添加新项”对话框，选择“Web 窗体”选项，删除留言页面文件名存为 DelMessage.aspx。

切换到设计视图，为 DelMessage.aspx 页面添加控件。从左侧工具箱数据组中拖出一个 XmlDataSource 控件和一个 GridView 控件。

右击 GridView 控件右上角的小三角按钮，在弹出的“GridView 任务”菜单中，选择“选择数据源”→“新建数据源”命令，打开“数据源配置向导”对话框，选择数据源类型为“XML 文件”，并在“为数据源指定 ID”文本框中输入 XmlDataSource1，如图 7-1 所示。

单击“确定”按钮，在随后弹出的配置数据源对话框中指定“数据文件”和数据“转换文件”，如图 7-2 所示。

单击 GridView 控件右上角的小三角按钮，在弹出的“GridView 任务”菜单中选择“添加新列”命令，打开“添加字段”对话框，选择字段类型为 TemplateField、页眉文本为“删除”，单击“确定”按钮。在“GridView 任务”对话框中，再选择“编辑模板”列，在 ItemTemplate 中添加一个 LinkButton 控件。单击 LinkButton 控件右上角的小三角按钮，在弹出的“LinkButton 任务”菜单中选择“编辑 DataBindings”命令，在打开的 LinkButton1 DataBindings 对话框中，选择要绑定的 CommandArgument 属性为 id，如图 7-5 所示。

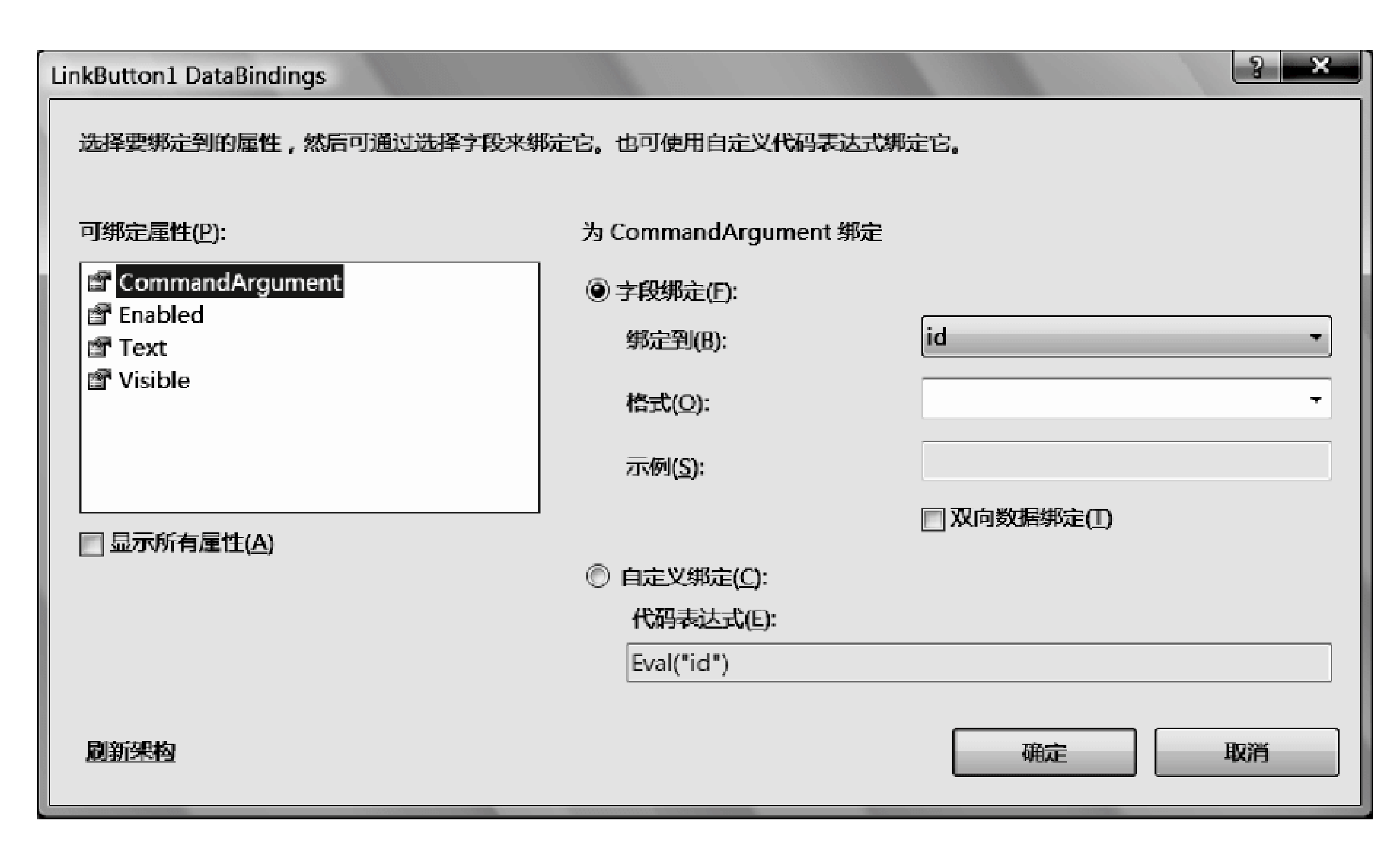


图 7-5 LinkButton1 DataBindings 对话框

DelMessage. aspx 删除页面主要代码如下：

```
<form id = "form1" runat = "server">
    <div>
            < asp: GridView  ID  =  " GridView1 "  runat  =  " server "  AllowPaging  =  " True "
AutoGenerateColumns = " False"  DataSourceID = " XmlDataSource2"  Width = ""  OnRowCommand =
"GridView1_RowCommand" CellPadding = "4" ForeColor = "#333333" GridLines = "None">
              <Columns>
                <asp:BoundField DataField = "id" HeaderText = "编号" SortExpression = "id" />
                <asp:BoundField DataField = "name" HeaderText = "姓名" SortExpression = "name" />
                <asp:BoundField DataField = "content" HeaderText = "发言内容" SortExpression =
"content" />
                <asp:BoundField DataField = "msgtime" HeaderText = "留言时间" SortExpression =
"msgtime" />
                <asp:TemplateField HeaderText = "删除">
                    <ItemTemplate>
                        <asp:LinkButton ID = "LinkButton1" runat = "server" CommandArgument =
'<%# Eval("id") %>' CommandName = "Del" CausesValidation = "false">删除</asp:LinkButton>
                    </ItemTemplate>
                </asp:TemplateField>
              </Columns>
        </asp:GridView>
        <asp:XmlDataSource ID = "XmlDataSource1" runat = "server" DataFile = "~/ XMLFile.
xml " TransformFile = "~/ XSLTFile.xsl ">
        </asp:XmlDataSource>
    </div>
</form>
```

右击“GridView 控件”，在弹出的快捷菜单中可以为 GridView 选择“自动套用格式”子菜单中的色彩方案。限于篇幅，这里省略了页面色彩方案部分的代码。

右击“GridView 控件”，在弹出的快捷菜单中选择“属性”命令，打开“属性”窗口，在 GridView1 的属性窗口中，单击“事件”按钮。双击触发 RowCommand 事件，在 DelMessage. aspx. cs 文件中编写 GridView1_RowCommand 事件代码如下：

```
…
using System.Xml;
public partial class DelMessage : System.Web.UI.Page
{
    protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e)
    {
        if (e.CommandName == "Del")
        {
            //初始化 XmlWrite 类
            XmlWrite WriteMessage = new XmlWrite();
            //获取选中的待删行的"id"值
            string passid = e.CommandArgument.ToString();
            WriteMessage.DeleNote(Server.MapPath("XMLFile.xml"),passid );
        }
    }
}
```

注意：DeleNote ()方法所需的第一个参数是 XMLFile. xml，第二个参数 passid 是单击“删除”按钮时，页面传递过来的待删留言的 id 值。id 参数值传递的设置参见图 7-5 所示的 LinkButton1 DataBindings 对话框，在其中设置 LinkButton 控件的 CommandArgument 属性值为 '<%＃ Eval("id") %>'。

DelMessage. aspx 删除留言页面运行效果如图 7-6 所示。

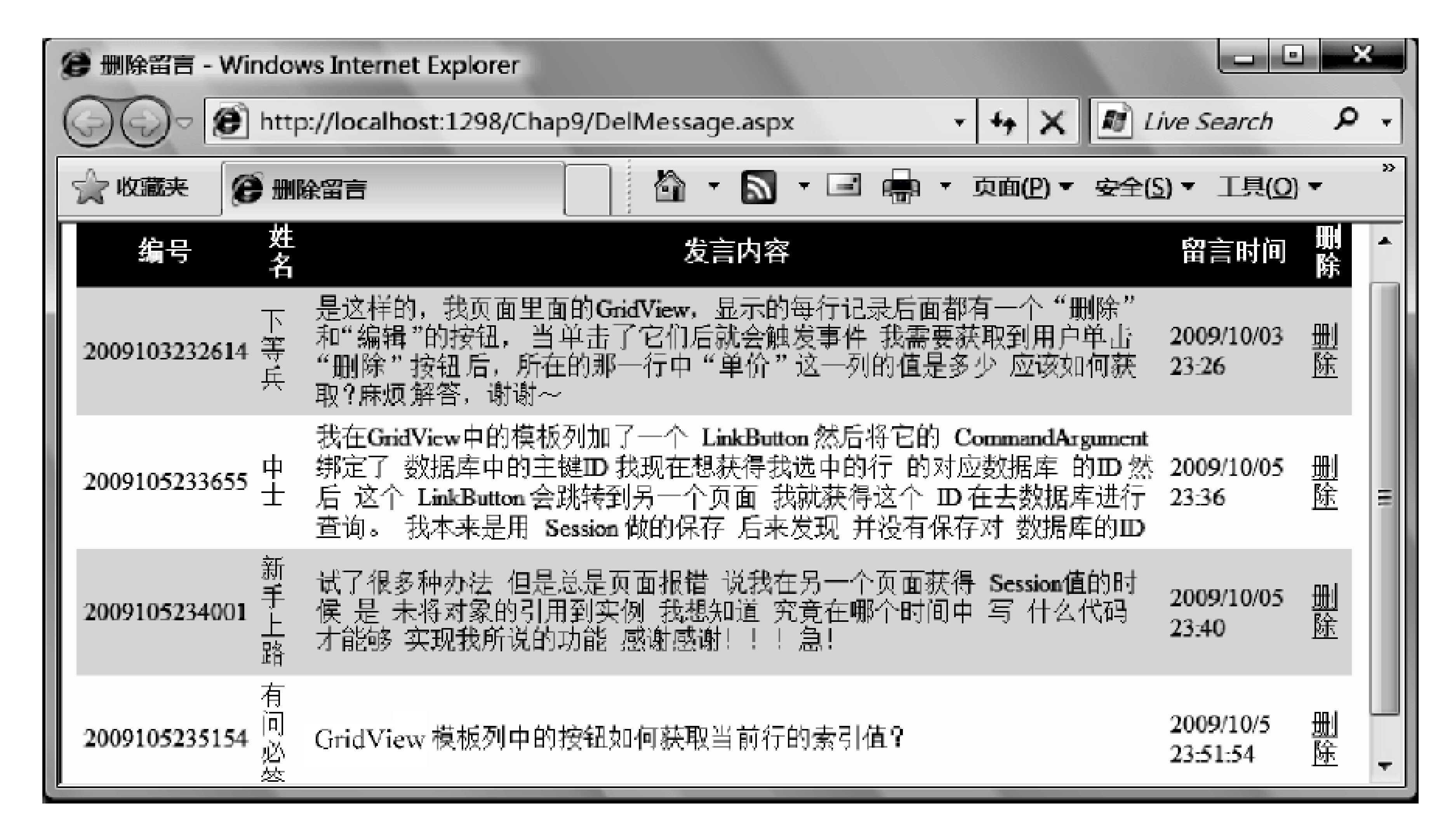


图 7-6 删除留言页面运行效果

7.1.7　小结

因为 XmlDataSource 数据源控件不支持 Insert、Update 以及 Delete 等方法，所以需要为访问 XML 封装、设计公共类，编写增、删、改、查 XML 数据的方法。

7.1.8　思考与练习

参照本书任务 9.1(9.1.8 小节)中 AspNetPager 分页控件的使用，给显示留言页面加上分页功能。

第 8 章　.NET Web 服务

任务 8.1　使用 Web 服务查询发布天气预报

任务目标

(1) 了解 Web 服务的概念以及应用前景。

(2) 掌握创建 Web 服务和使用 Web 服务的方法。

8.1.1　Web 服务概述

Web 服务提供了一种可以在不同编程语言、不同体系架构以及不同操作系统的 Web 应用程序之间交互通信的平台。Web 服务是建立在 XML、SOAP (Simple Object Access Protocol,简单对象访问协议)、WSDL(Web Services Definition Language,Web 服务定义语言)和 UDDI(Universal Description Discovery Integration,统一描述、查找、集成)等 Web 行业开放标准的基础上的,允许任何人开发或者使用它们。

Web Service 通过 SOAP 协议来实现 XML 格式数据的传输;WSDL 是一个用来描述 Web Service 访问方式的 XML 文档;UDDI 是用来注册发布 Web Service 和搜索发现 Web Service 信息的一个标准。

有越来越多的商务应用、金融财务事务乃至公共服务使用 Web 服务模式,有人预测 Web 服务将成为未来动态商务 Web 的主流技术。

8.1.2　一个简单的 Web 服务实例

下面通过 Visual Studio 提供的 Web 服务样例来说明创建一个 Web 服务和使用 Web 服务的过程。

创建一个 Web 服务的过程如下。

(1) 运行 Visual Studio 2005。选择菜单“文件”→“新建”→“网站”命令,在弹出的“新建网站”对话框中选择“ASP.NET Web 服务”模板,注意这是一个提供 Web 服务的网站模板,给网站命名为 WebService1。

(2) 单击“确定”按钮,在“解决方案资源管理器”面板中,可以看到用来实现 Web 服务的 Service. asmx 文件和 Service. cs 文件,如图 8-1 所示。在 Service. cs 后置代码文件中,有自动生成的代码如下:



图 8-1　Web 服务网站 WebService1

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;

[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service ()
    {
        //如果使用设计的组件,请取消注释以下行
        //InitializeComponent();
    }

    [WebMethod]
    public string HelloWorld() {
        return "Hello World";
    }
}
```

在这里,模板定义了一个 public 类 Service,它继承自 System.Web.Services.WebService 类。Service 类中已经定义了一个 Web 服务方法 HelloWorld(),调用该方法将返回一个 "Hello World"字符串。

为区别于其他网站的 Web 服务,可以修改 Web 服务的默认命名空间,比如可以改为 Namespace = "http://www.0931.org/"。

打开 Service.asmx 文件,其中只有一行代码,照录如下:

```
<%@ WebService Language = "C#" CodeBehind = "~/App_Code/Service.cs" Class = "Service" %>
```

(3) 在"解决方案资源管理器"面板中右击 E:\WebService1\,在弹出的快捷菜单中选择"生成网站"命令。网站生成后,右击 Service.asmx 文件,在弹出的快捷菜单中选择"在浏览器中查看"命令。Service.asmx 运行结果如图 8-2 所示。此 Web 服务页面(Service.asmx)的 URL:http://localhost:49690/WebService1/Service.asmx 可记下备用。至此,一个 Web 服务创建完毕。



图 8-2 查看 WebService1 提供的 Web 服务

(4) 单击 HelloWorld 超链接,可以看到此 Web 服务的测试调用窗口,如图 8-3 所示。

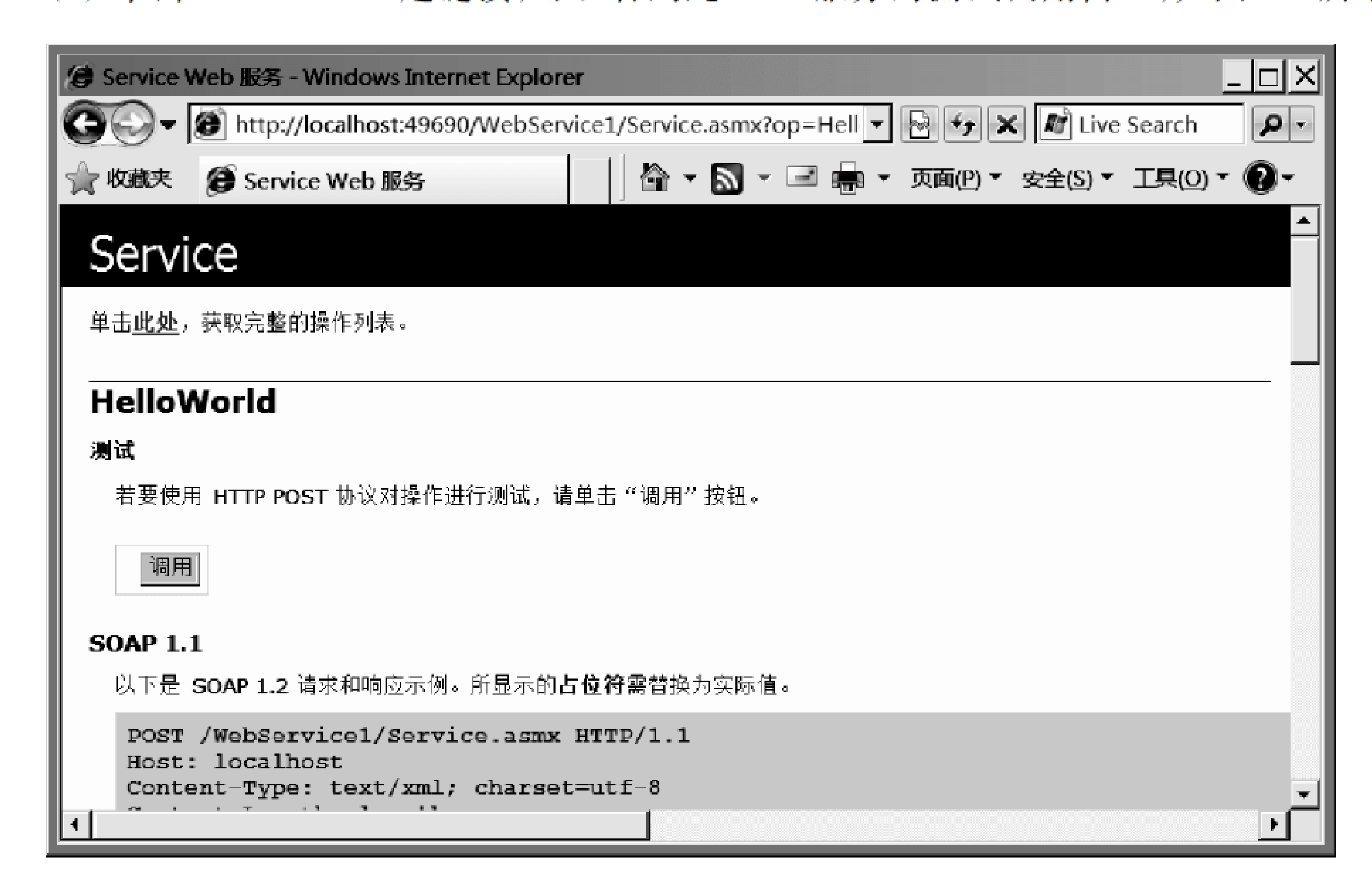


图 8-3　Web 服务的测试调用窗口

单击"调用"按钮,可以对此 Web 服务进行调用测试,测试结果如图 8-4 所示。

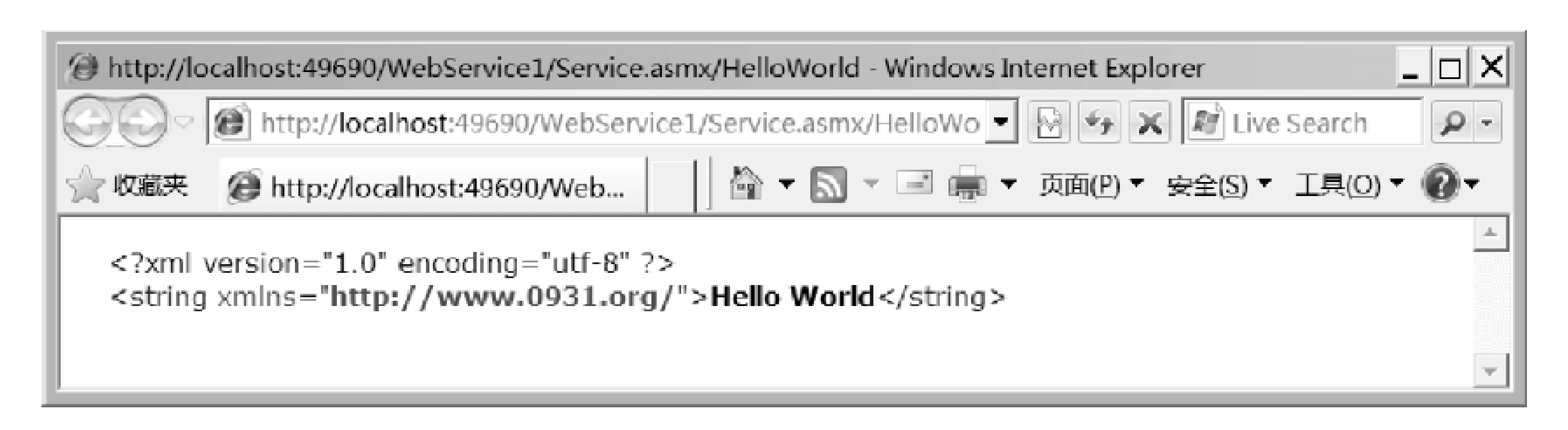


图 8-4　测试返回结果

以下是使用 Web 服务的过程,说明如何在另一服务器或者另一网站上使用上述 Web 服务。

(1) 创建使用 Web 服务的网站。运行 Visual Studio 2005,选择菜单"文件"→"新建"→"网站"命令,打开"新建网站"对话框,选择"ASP.NET 网站"选项,给网站命名为 WebService2。

(2) 在"解决方案资源管理器"面板中右击 WebService2,在弹出的快捷菜单中选择"添加 Web 引用"命令。在图 8-5 所示的"添加 Web 引用"对话框的 URL 地址栏中,输入提供 Web 服务的 Service.asmx 页面的 URL:http://localhost:49690/WebService1/Service.asmx。此 URL 是之前记下备用的,是 WebService1 网站提供 Web 服务的 Service.asmx 页面的 URL。也可以在图 8-2 所示的 IE 浏览器的地址栏中看到。

注意:发布 Web 服务的页面 URL 的端口号是随机发生变化的。另外,如果提供

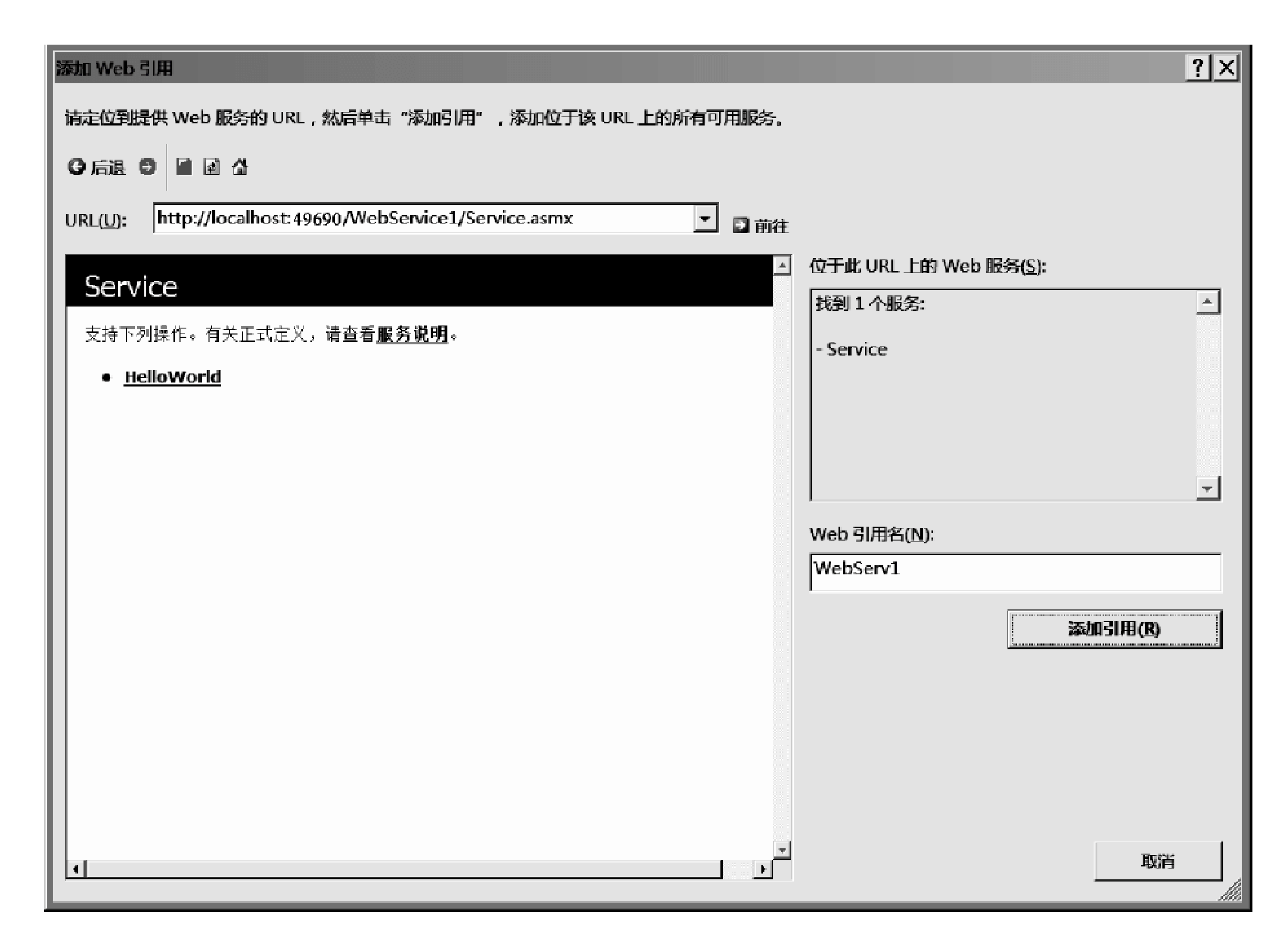


图 8-5　“添加 Web 引用”对话框

Web 服务的网站在另一服务器上，则 localhost 应当是别的服务器的域名或 IP 地址。

单击“前往”按钮，待找到位于此 URL 上的 Web 服务后，输入 Web 引用名，这里命名为 WebServ1。

(3) 单击“添加 Web 引用”对话框中的“添加引用”按钮，一个以 WebServ1 命名的 SOAP 代理类自动生成，如图 8-6 所示。其中 Service. wsdl(Web Service Description Language，WSDL)是一个 XML 格式的描述文档，说明此 Web 服务中定义的类、方法以及所需参数等。

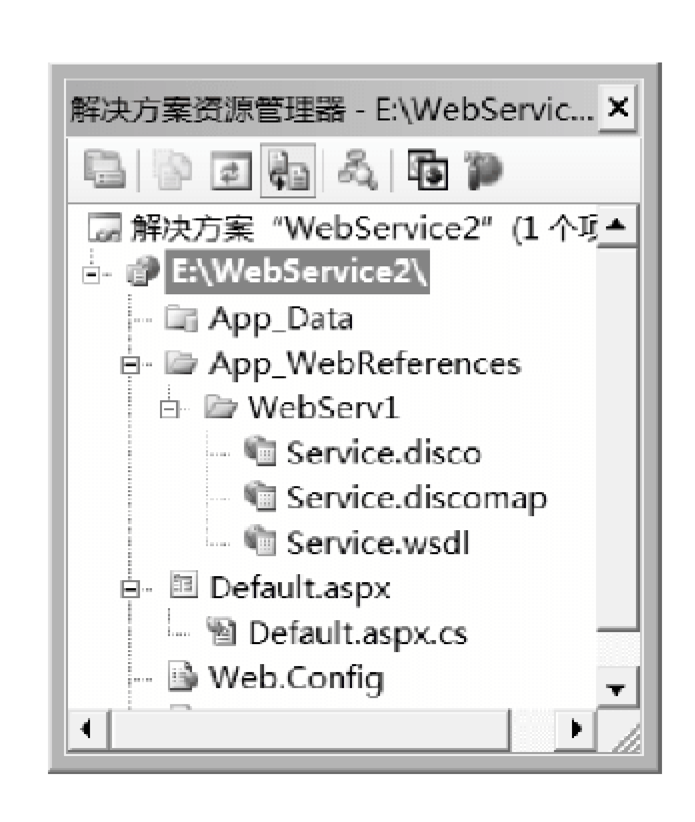


图 8-6　生成 WebServ1 代理类

在站点应用程序配置文件 Web. Config 的<appSettings>节中，可以看到自动增加的代码如下：

```
<configuration>
   <appSettings>
         < add key = " WebServ1. Service" value = " http://localhost: 49690/WebService1/
Service.asmx"/>
   </appSettings>
      ...
</configuration>
```

注意：当发布 Web 服务的页面 URL 或端口号发生变化时，要在此修改 value 的值。

至此，站点 A 上的 Web 服务成为站点 B 上的一个内置类，站点 B 可通过此(内置的)代理类向 Web 服务发送请求并返回结果。

(4) 在 Default.aspx 页面添加一个 Label 控件、一个 Button 控件，用来调用 Web 服务和显示调用结果。Default.aspx 页面主要代码如下：

```
<html xmlns = "http://www.w3.org/1999/xhtml" >
<head runat = "server"><title>调用 Web 服务</title></head>
<body>
    <form id = "form1" runat = "server">
    <div>
        <asp:Label ID = "Label1" runat = "server" Text = "显示返回结果"></asp:Label>
        <asp:Button ID = "Button1" runat = "server" Text = "获取 Web 服务" OnClick =
"Button1_Click"/>
    </div>
    </form>
</body>
</html>
```

在 Default.aspx.cs 的 Button1_Click 中编写如下代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
     //创建代理类对象实例并调用实例方法
      WebServ1.Service simpleWebServ = new WebServ1.
Service();
     string strResult = simpleWebServ.HelloWorld();
     //返回结果赋值给 Label1
     Label1.Text = strResult;
}
```

(5) 运行 Default.aspx，单击“获取 Web 服务”按钮，在 Label1 中将显示调用此 Web 服务的结果，如图 8-7 所示。



图 8-7 调用 Web 服务结果

8.1.3 一个返回 DataSet 对象的电话区号查询 Web 服务实例

下面开发一个提供电话区号查询的 Web 服务：输入城市名称，即可返回一个对应的电话区号。以此为例，进一步说明一个实用的 Web Service 的创建和使用过程。

此查询类的 Web 服务返回的是一个 DataSet 对象，提供此 Web Service 的站点需要有后台数据源支持，这里使用 SQL Server 2005 数据库。数据库表 telecode 的简单设计如表 8-1 所示。

表 8-1 telecode 表结构

字　段　名	字段类型	主键及字段属性
id	int	主键，标识增量 1、标识种子 1
CityName	nchar(10)	城市名
TelephoneCode	nchar(10)	电话号码

Web Service 的创建过程如下。

(1) 运行 Visual Studio 2005。在“解决方案资源管理器”面板中，右击 Web 服务网站 E:\WebService1\，在弹出的快捷菜单中选择“添加”→“新建项”命令，在弹出的“添加新项-E:\WebService1”对话框中，选择“SQL 数据库”，创建一个名为 webServInfo.mdf 的数据库及数据库表 telecode。数据库及表的创建步骤参照 5.1.3 小节。

选择将 webServInfo.mdf 放在 App_Data 文件夹中。在“服务器资源管理器”面板中右击“数据连接”，在弹出的快捷菜单中选择“添加连接”命令，在弹出的“添加连接”对话框中，选择数据源为“Microsoft SQL Server 数据库文件 (SqlClient)”；单击“浏览”按钮，选择数据库文件名，这里是：E:\ WebService1\App_Data\webServInfo.mdf。在连接数据库时，可以使用 SQL Server 身份验证，输入数据库用户名和密码，也可以使用 Windows 身份验证，这里使用 Windows 身份验证，单击“测试连接”按钮，测试是否连接成功。

在 Web.config 配置文件的<appSettings>节中添加如下数据库连接字符串代码。

```
<configuration>
<appSettings/>
    <connectionStrings>
          <add name = "webServInfoConnectionString" connectionString = "Data Source =
.\SQLEXPRESS;AttachDbFilename = |DataDirectory|\webServInfo.mdf;Integrated Security =
True;User Instance = True" providerName = "System.Data.SqlClient" />
     </connectionStrings>
…
 </configuration>
```

如何在 Web.config 配置文件中自动生成数据库连接字符串以及 Web.Config 文件中数据库连接字符串获取的详细说明，参见本书 9.1.3 小节。

(2) 在 Service.cs 文件的 public class Service 类定义中增加一个 getTelephoneCode WebMethod。Service.cs 的实现代码如下：

```
using System;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Data;
using System.Data.SqlClient;

[WebService(Namespace = "http://www.0931.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
public class Service : System.Web.Services.WebService
{
    public Service()
    {
        //如果使用设计的组件,请取消注释以下行
        //InitializeComponent();
    }

    [WebMethod]
```

```
    public string HelloWorld()
    {
        return "Hello World";
    }

    [WebMethod]
    public DataSet getTelephoneCode(string cityName)
    {
        // 创建 SqlConnection 对象
        SqlConnection conn = null;
        //设置数据在内存中的缓存 DataSet
        DataSet ds = null;

        try
        {
            //从 Web.Config 中获取数据库连接字符串
            string strConn =
System.Configuration.ConfigurationManager.ConnectionStrings["webServInfoConnectionString"].
ConnectionString;

            //准备 SQL 语句(模糊查询)
            string strSql = "Select * From telecode Where CityName LIKE'%" + cityName + "%'";

            //初始化 SqlConnection 类的新实例.使用 using 语句可以及时释放资源
            using (conn = new SqlConnection(strConn))
            {
              //初始化 DataSet 类的新实例
                ds = new DataSet();
                conn.Open();//打开数据库连接

                //定义一个 Adapter 执行 SQLServer 命令并保存数据
                SqlDataAdapter da = new SqlDataAdapter(strSql, conn);

                da.Fill(ds); //给 DataSet 填充数据
                return (ds); //返回 DataSet
            }
        }
        catch (SqlException)
        {
            return ds = null;
        }
    }
}
```

(3) 在“解决方案资源管理器”面板中右击 E:\WebService1\，在弹出的快捷菜单中选择“生成网站”命令。网站生成后，右击 Service.asmx 文件，在弹出的快捷菜单中选择“在浏览器中查看”命令。Service.asmx 运行结果如图 8-8 所示。此 Web 服务页面的 URL：http://localhost:49453/WebService1/Service.asmx 可记下备用。至此，一个 Web

服务创建完毕。

在图 8-8 所示的“Service Web 服务”页面中单击 getTelephoneCode 超链接，可以看到此 Web 服务的测试调用窗口，如图 8-9 所示。

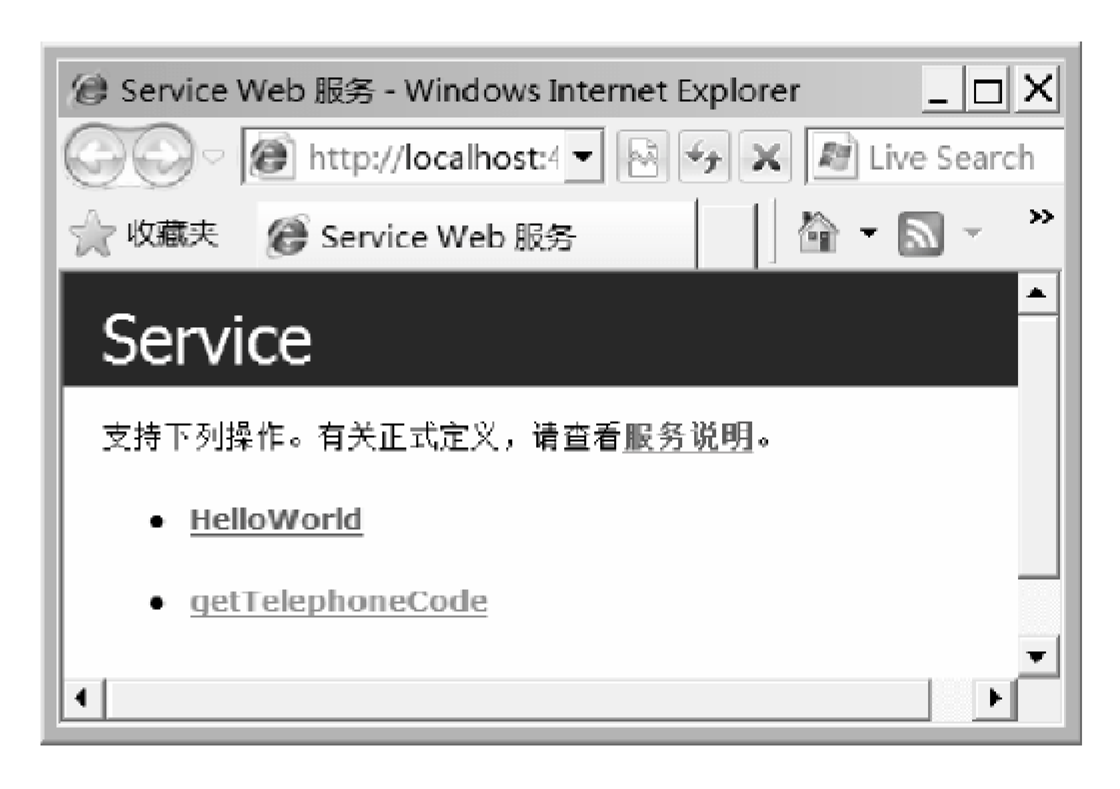


图 8-8 Service. asmx 运行结果

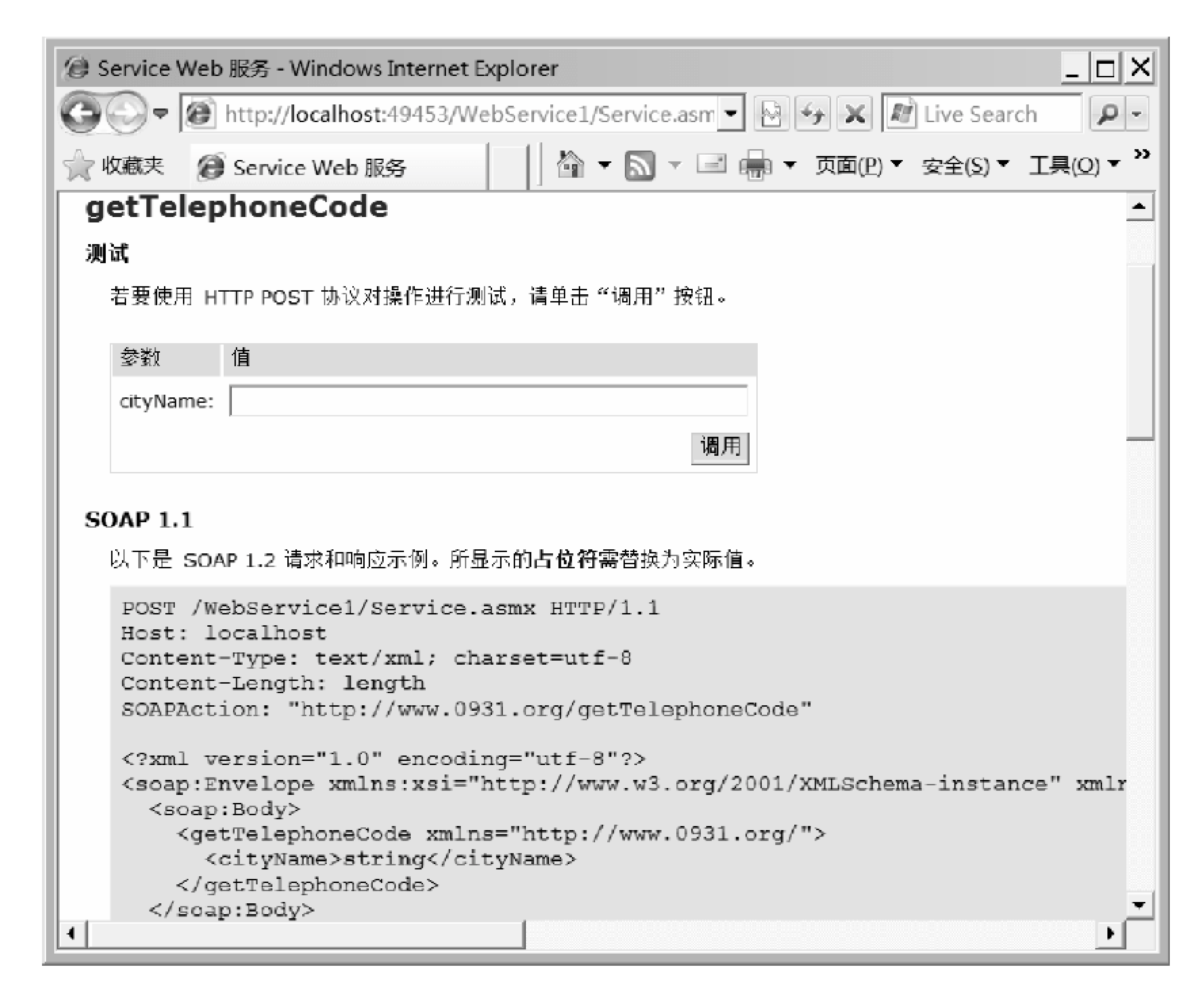


图 8-9 Web 服务的测试调用窗口

以下是在另一服务器网站上使用上述网站提供的 Web 服务的过程。

(1) 运行 Visual Studio 2005。打开网站 WebService2。

(2) 在“解决方案资源管理器”面板中右击 E:\WebService2\，在弹出的快捷菜单中选择“添加 Web 引用”命令。在打开的“添加 Web 引用”对话框的 URL 地址栏中，输入提

供 Web 服务的 Service. asmx 页面的 URL：http://localhost:49453/WebService1/Service. asmx。

单击"前往"按钮，待找到位于此 URL 上的 Web 服务后，输入 Web 引用名，这里命名为 TeleCode。

(3) 单击"添加引用"按钮，在"解决方案资源管理器"面板中，可以看到一个以 TeleCode 命名的 SOAP 代理类自动生成，如图 8-10 所示。



图 8-10　生成的 TeleCode 代理类

在站点配置文件 Web. Config 的<appSettings>节中，可以看到自动增加了一行代码如下：

```
<configuration>
    <appSettings>
        <add key = " WebServ1. Service" value = " http://localhost: 49690/WebService1/
Service. asmx"/>
        <add key = " TeleCode. Service" value = " http://localhost: 49453/WebService1/
Service. asmx"/>
    </appSettings>
    …
</configuration>
```

至此，站点 1 上的 Web 服务成为站点 2 上的一个内置类，站点 2 可通过此(内置的)Telecode 代理类向 Web 服务发送请求并返回结果。

(4) 在 WebService2 站点下，新建一 Default2. aspx 页面，用来充当输入查询的用户界面。在页面上添加一个 Label 控件、一个 TextBox 控件、一个 Button 控件和一个 GridView 控件。

Default2. aspx 主要代码如下：

```
<head runat = "server"><title>Web Service 查询电话区号</title></head><body>
<form id = "form1" runat = "server">
    <div>
        <asp:Label ID = "Label1" runat = "server" Text = "输入城市名"></asp:Label>
        <asp:TextBox ID = "TextBox1" runat = "server"></asp:TextBox>
        <asp:Button ID = " Button1 " runat = " server " Text = " 查 询 电 话 区 号 " OnClick =
"Button1_Click" />
        <asp:GridView ID = "GridView1" runat = "server" AllowPaging = "true" PageSize = "6">
        </asp:GridView>
    </div>
</form></body>
```

在 Default2. aspx. cs 文件的 Button1_Click 中编写代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
        if (this.TextBox1.Text ! = "")
        {
                string city_name = this.TextBox1.Text;
                //创建代理类实例
```

```
            TeleCode.Service TeleCodeWebServ = new TeleCode.Service();
            //创建 DataSet 类实例
            DataSet ds = new DataSet();

            //调用代理类的实例方法,返回 DataSet 对象
            ds = TeleCodeWebServ.getTelephoneCode(city_name);
            //给 GridView 控件指定数据源
            this.GridView1.DataSource = ds;
            this.GridView1.DataBind();//执行绑定
        }
        else
        {
            Response.Write("<script>alert('查询区号请输入城市名称!')</script>");
        }
    }
```

(5) 运行 Default2.aspx,输入省份或城市名,单击"查询电话区号"按钮,调用 Web 服务结果如图 8-11、图 8-12 所示。



图 8-11 调用 Web 服务结果(1)



图 8-12 调用 Web 服务结果(2)

8.1.4 使用 Web 服务查询发布天气预报

在 Internet 上可以检索到一些比较稳定的、免费提供天气预报 Web 服务的网站,比如 http://www.ayandy.com/(China)、http://www.xmethods.net/(United States)等。

在 IE 中打开 http://www.ayandy.com/service.asmx,该网站 Web 服务测试调用窗口如图 8-13 所示。注意查看 Web 方法 getWeatherbyCityName 的输入参数(城市中文名称,今天/明天/后天)和返回数据(一个有 7 个元素的一维数组,从 String a[1]～ a[6]分别代表城市、天气、温度、风向、日期和天气图标地址)。

下面简单地说明如何使用 http://www.ayandy.com/提供的 Web 服务在自己的网站上查询发布天气预报。

(1) 运行 Visual Studio 2005。打开网站 WebService2。

(2) 在"解决方案资源管理器"面板中右击 E:\WebService2\,在弹出的快捷菜单中

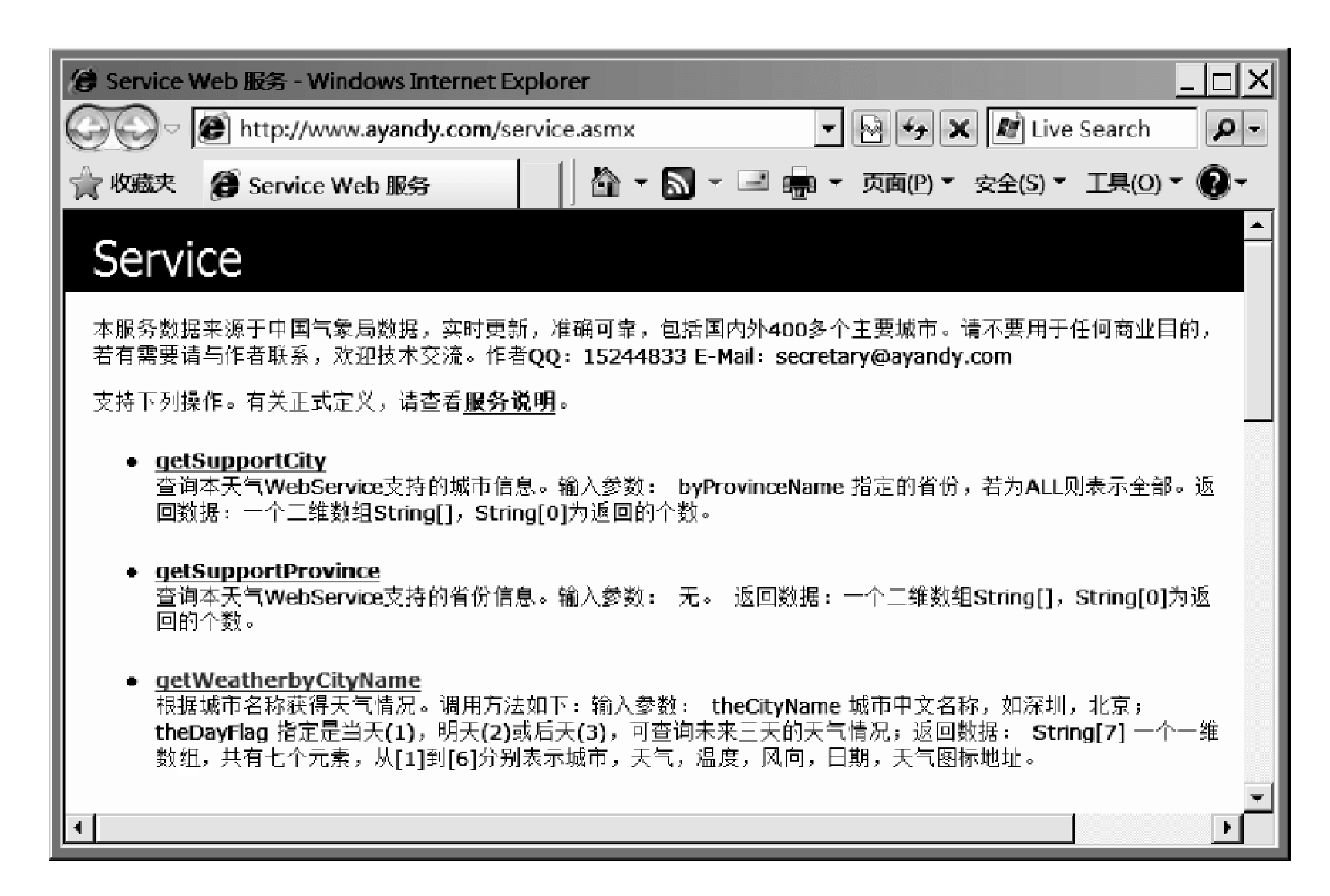


图 8-13　http://www.ayandy.com/Web 服务的测试调用窗口

选择“添加 Web 引用”命令。在打开的“添加 Web 引用”对话框的 URL 地址栏中，输入提供 Web 服务的 Service.asmx 页面的 URL：http://www.ayandy.com/service.asmx。注意需要连接上 Internet。

单击“前往”按钮，待找到位于此 URL 上的 Web 服务后，输入 Web 引用名，这里命名为 ayandy。

(3) 单击“添加引用”按钮，在“解决方案资源管理器”面板中，可以看到一个以 ayandy 命名的 SOAP 代理类自动生成。

在站点配置文件 Web.Config 的<appSettings>节中，可以看到又自动增加了一行代码如下：

```
<configuration>
    <appSettings>
        <add key = " WebServ1. Service" value = " http://localhost: 49690/WebService1/
Service.asmx"/>
        <add key = " TeleCode. Service" value = " http://localhost: 49453/WebService1/
Service.asmx"/>
        <add key = "ayandy.service" value = "http://www.ayandy.com/service.asmx"/>
    </appSettings>
     …
</configuration>
```

下面就可以通过 ayandy 代理类向远程 Web 服务发送请求并返回结果了。

(4) 在 WebService2 的 Default3.aspx 页面上添加一个 Image 控件用来显示代表阴晴雨雪的图片；添加 5 个 Label 控件分别用来显示城市、日期、天气、温度和风向信息；添

加一个 TextBox 控件、一个 Button 控件用来输入并提交查询信息。

Default3. aspx 页面主要代码如下：

```
<form id = "form1" runat = "server">
    <div>
        <asp:Image ID = "Image1" runat = "server" Visible = "False" />
        <asp:Label ID = "Label1" runat = "server" Text = "Label"></asp:Label>
        <asp:Label ID = "Label2" runat = "server" Text = "Label"></asp:Label>
        <asp:Label ID = "Label3" runat = "server" Text = "Label"></asp:Label>
        <asp:Label ID = "Label4" runat = "server" Text = "Label"></asp:Label>
        <asp:Label ID = "Label5" runat = "server" Text = "Label"></asp:Label><br />
        <div>
            输入城市名<asp:TextBox ID = "TextBox1" runat = "server"></asp:TextBox>
                    < asp: Button ID = " Button1" runat = " server" Text = " 查 询 天 气"
OnClick = "Button1_Click" /></div>
    </div>
</form>
```

在 Default3. aspx. cs 的 Page_Load 事件上编写代码发布本地天气预报信息、在 Button1_Click 事件中编写代码处理页面提交的查询信息。

Default3. aspx. cs 的实现代码如下：

```
public partial class Default2 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            //创建代理类对象实例
            ayandy.Service WeatherWebServ = new ayandy.Service();

            //调用 getWeatherbyCityName 实例方法并返回数据()
            string[] WeatherForecastToday =
            WeatherWebServ.getWeatherbyCityName("台州", ayandy.theDayFlagEnum.Today);

            //将返回数据绑定到页面显示控件
            Image1.Visible = true;
            Image1.ImageUrl = WeatherForecastToday[6]; //天气图标
            Label1.Text = WeatherForecastToday[1]; //城市
            Label2.Text = WeatherForecastToday[5]; //日期
            Label3.Text = WeatherForecastToday[2]; //天气
            Label4.Text = WeatherForecastToday[3]; //温度
            Label5.Text = WeatherForecastToday[4]; //风向
        }
    }
}
    protected void Button1_Click(object sender, EventArgs e)
    {
        ayandy.Service WeatherWebServ = new ayandy.Service();
```

```
        string[] WeatherQueryToday =
        WeatherWebServ.getWeatherbyCityName(TextBox1.Text, ayandy.theDayFlagEnum.Today);

        Image1.ImageUrl = WeatherQueryToday[6];
        Label1.Text = WeatherQueryToday[1];
        Label2.Text = WeatherQueryToday[5];
        Label3.Text = WeatherQueryToday[2];
        Label4.Text = WeatherQueryToday[3];
        Label5.Text = WeatherQueryToday[4];
    }
```

TextBox1.Text 为用户输入的、要查询天气的城市名称。运行 Default3.aspx，运行效果如图 8-14、图 8-15 所示。



图 8-14　天气预报 Web 服务(1)



图 8-15　天气预报 Web 服务(2)

8.1.5　小结

Web Service 可以返回字符串、整数、逻辑值、数组、日期等基本类型数据，也可以返回一个类甚至 DataSet 对象。DataSet 是数据库返回的数据在服务端内存中的缓存，通过 Web Service 发送 DataSet，服务端无须再连接数据库。在互联网上可以找到越来越多的

返回 DataSet 的 Web Service 并使用它们，也可以创建并注册类似的 Web Service，让更多的互联网用户使用它。

8.1.6 思考与练习

1. 试着为银行开发一个可提供当日汇率查询的 Web 服务。
2. 为 0931 网站首页添加天气预报 Web 服务消息。

第 9 章　网站部署与安全性配置

任务 9.1　一个三层架构的博客网站

任务目标

（1）实现一个可以浏览、发表日志和评论，提供后台管理的博客网站。

（2）了解网站安全性配置策略的过程。

9.1.1　Web.Config 文件概述

ASP.NET 提供了一个基于 XML 的、强大而简单可行的配置系统，它是层次架构的。一个应用程序级别的 Web.config 配置文件就是放置在应用程序根目录下的一个 XML 文件。

在 C:\Windows\Microsoft.NET\Framework\v2.0.50727(versionNumber)\CONFIG 下有一个 Machine.config 文件，它是 Web 服务器上所有.NET 应用程序的配置文件，包含应用程序的常用配置信息。所有 Web.config 都继承自 Machine.config。

设置 Web.config 中的配置信息可以实现网站的安全控制，比如可以设置身份验证及类型；可以创建用户和角色（用户组）；可以创建对应用程序各个部分的访问规则及页面授权；可以选择数据的连接方式和保存、设置加密或不加密的数据库连接字符串；可以设置应用程序的调试与跟踪、设置自定义错误页；等等。

在本章的数个任务中，试图通过创建并配置一个三层架构的、名为 MicroBlog 的博客（日志）网站来说明和实践应用程序安全性配置策略的过程。

9.1.2　系统三层结构与功能分析

下面分析说明 MicroBlog 系统的功能、结构以及安全性配置策略。

（1）MicroBlog 解决方案树形目录结构如图 9-1 所示，其中在站点（Web 表示层）下的主要文件夹和文件如下。

- App_Data 文件夹
  userLog.mdf　在 SQL Server 2005 系统下创建的数据库文件。
- Login 文件夹　实现用户登录、新用户注册、退出登录页面等。
- Admin 文件夹　实现用户、日志文章、评论等删改的后台管理页面。
- Web.config　站点应用程序配置文件。

此外，还有放在站点根目录下的一些浏览日志、显示评论、发表日志、发表评论、显示用户列表等的 Web 页面文件。

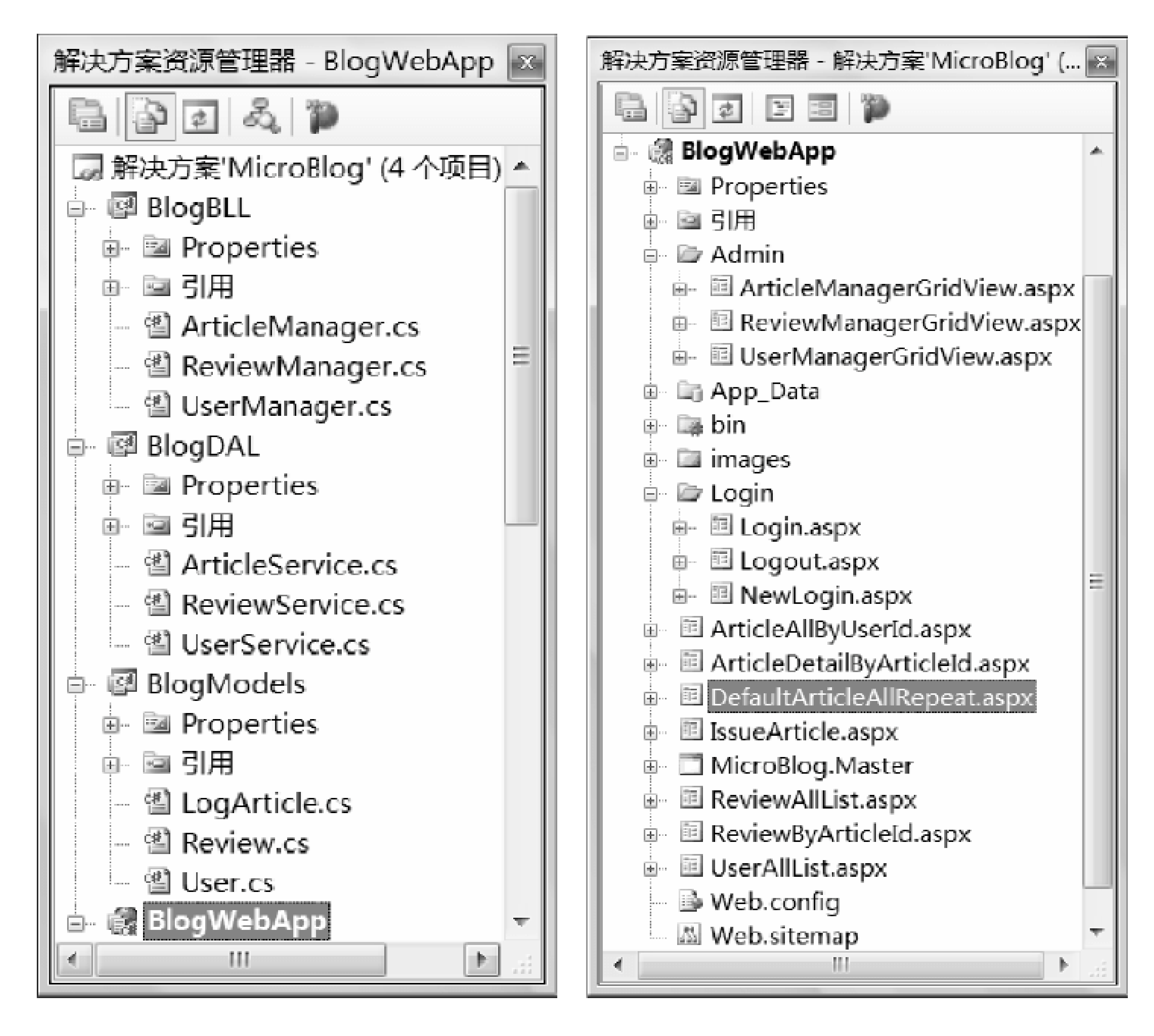


图 9-1　MicroBlog 解决方案树形目录

网站首页 DefaultArticleAllRepeat.aspx 是一个可以浏览显示全部日志文章的页面，如图 9-2 所示。在首页中间部分，单击“查看全文”和“评论”超链接，可以看到一篇文章的

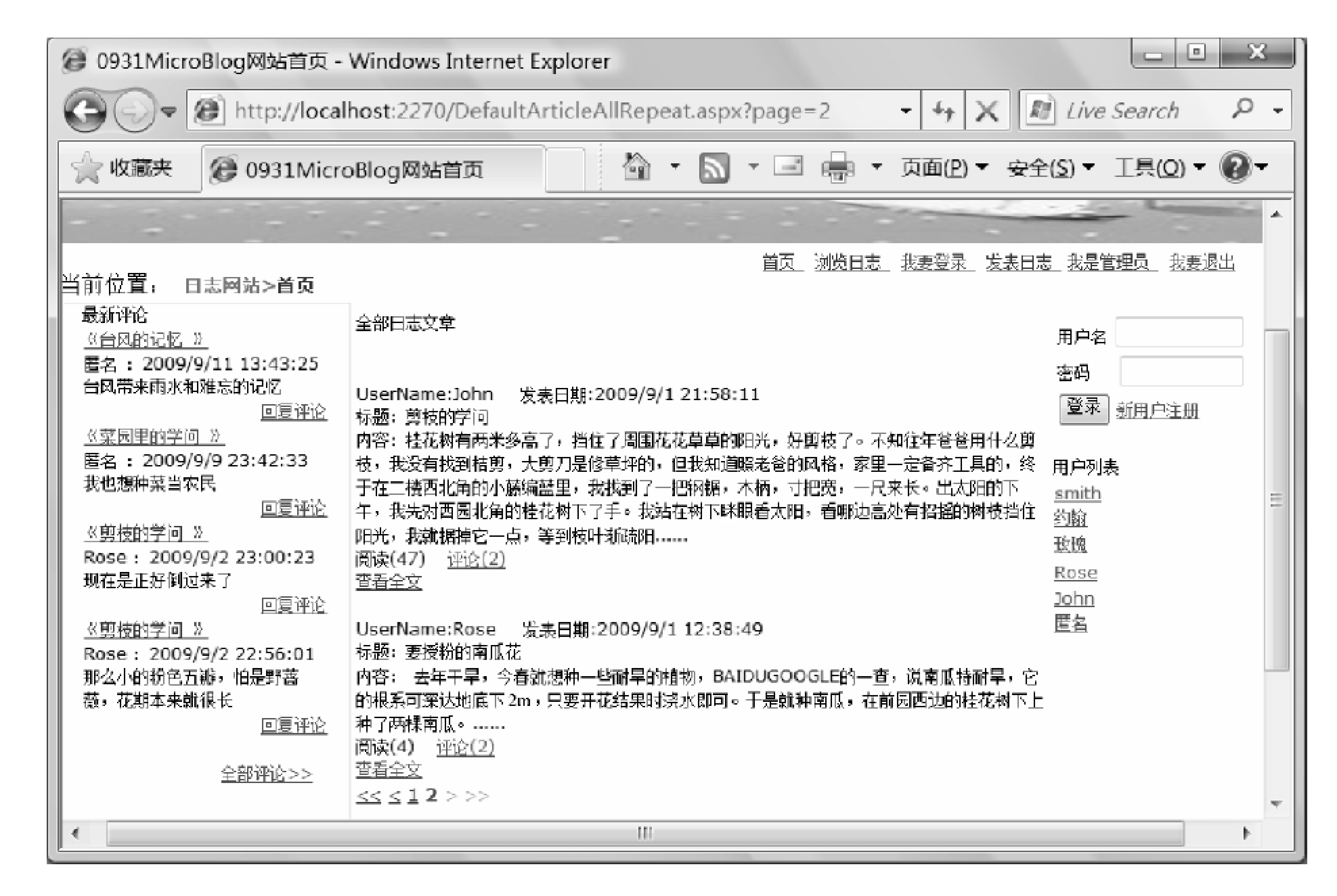


图 9-2　MicroBlog 站点首页

详细内容和全部评论。在首页右侧的“用户列表”下单击用户名超链接,可以查看一个用户的全部文章。在首页左侧的“最新评论”栏目,可以看到所有日志文章的最新评论和所有评论。在 Web 应用程序的各个页面中设置有导航栏,提供“首页”、“我要登录”、“浏览日志”、“发表日志”、“我是管理员”、“我要退出”等超链接导航。

请参照本书任务 5.1,在 Visual Studio 2005 下,创建一个名为 MicroBlog 的解决方案,搭建基于三层结构的系统框架以及各层次之间的依赖关系。并在 Web 表示层下按照上述目录结构部署文件和文件夹。

(2) 将数据库连接字符串部署在应用程序的 Web. config 文件中。

为简化应用程序编程,方便移植与维护,将数据库连接字符串设置在应用程序的 Web. config 文件中。注意当应用程序访问数据库时是从 Web. config 中获取数据库连接字符串的。

(3) 设置身份验证和页面授权。

对于访问站点的不同用户,比如未经登录验证的匿名用户、经过登录验证的认证用户、管理员账号用户等,分别授予允许访问或拒绝访问特定页面资源的权利;另外,还要对 Admin 文件夹的访问权限进行设置等。这些问题在下一任务中通过配置 Web. config 能全部完成。

在任务 9.1 中,只通过 Session 对象实现页面间的状态保持和数据传递:允许所有用户访问首页、浏览全部日志文章并发表评论、查看评论;允许所有用户访问 Login 文件夹并且可以登录、新用户注册或退出登录。当用户请求发表日志页面时,将提示“请先登录!”,而且只允许已注册的用户登录,当用户登录并通过数据库验证后,就可以发表日志了。

9.1.3 SQL Server 数据库的设计与连接

(1) 运行 Visual Studio 2005。在“解决方案资源管理器”面板中,右击 MicroBlog 项目的 Web 层 BlogWebApp,在弹出的快捷菜单中选择“添加”→“新建项”命令,在弹出的“添加新项- BlogWebApp”对话框中,选择“SQL 数据库”选项,创建一个名为 userlog.mdf 的数据库,其主要数据库表有:userinfo(用户信息表)、article(日志文章表)、review(评论表)。表结构设计如表 9-1～表 9-3 所示,数据库表间关系如图 9-3 所示。数据库及表的创建步骤参照 5.1.3 小节。

表 9-1 用户信息表(userinfo)

字段名称	类 型	说 明
user_id	int	用户 ID,主键。是标识,标识增量 1,标识种子 1
user_name	nchar(10)	用户登录名
user_pw	nchar(10)	用户登录密码
QQ	nchar(10)	用户 QQ
login_date	datetime	用户注册时间

表 9-2　日志文章表(article)

字段名称	类　型	说　明
arti_id	int	文章 ID,主键,自动加 1
user_id	int	用户 ID,外键,关联表 userinfo
arti_title	nchar(50)	文章标题
arti_text	nvarchar(2000)	文章内容
arti_click	int	文章阅读次数
submit_date	datetime	文章提交时间

表 9-3　评论表(review)

字段名称	类　型	说　明
revi_id	int	评论 ID,主键,自动加 1
arti_id	int	文章 ID,外键,关联表 article
user_id	int	用户 ID,外键,关联表 userinfo
revi_text	nvarchar(1000)	评论内容
submit_date	datetime	评论提交时间

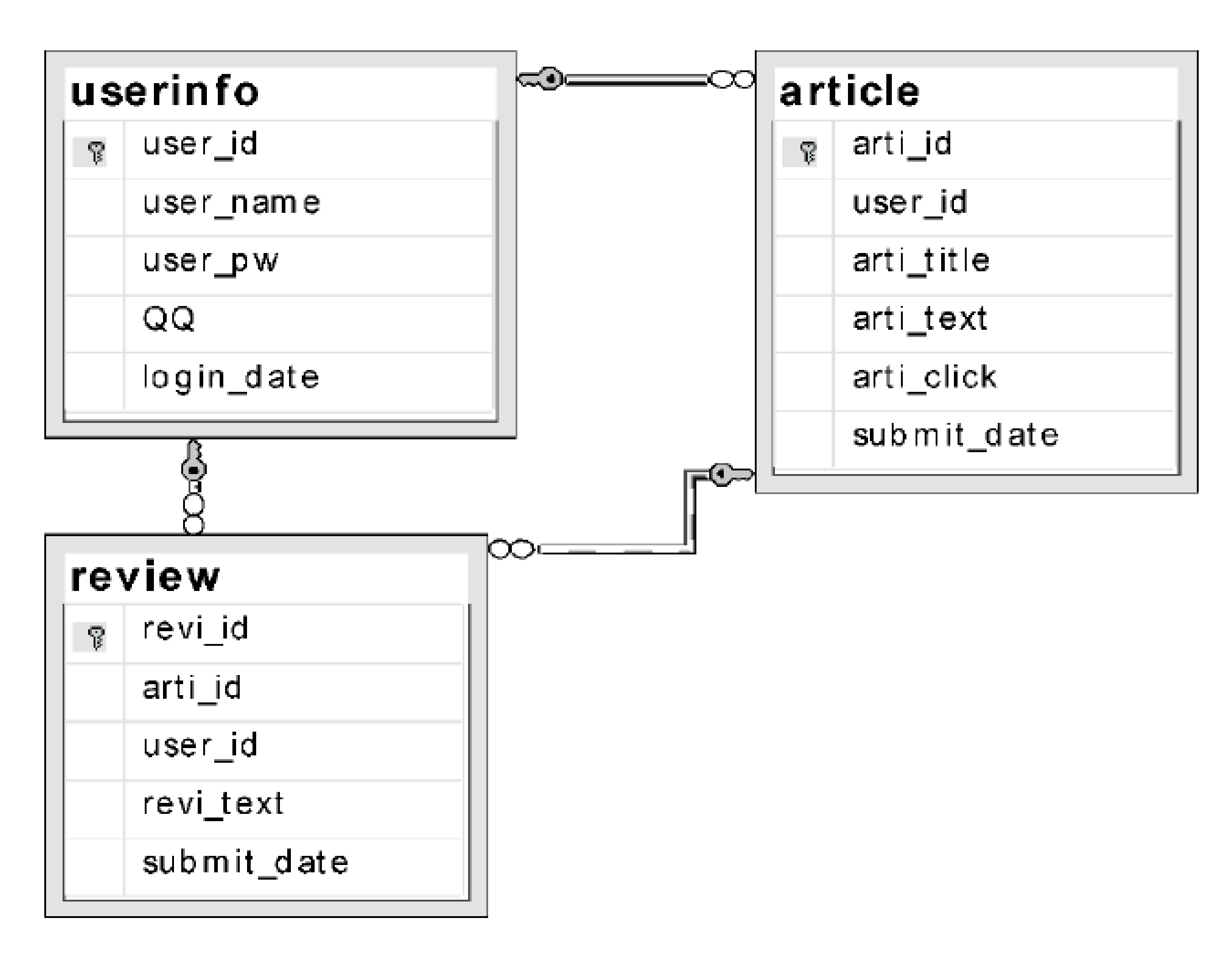


图 9-3　数据库表间关系

(2) 在 Visual Studio 2005 中连接 userlog 数据库。

选择将 userlog.mdf 放在 App_Data 文件夹中。在“服务器资源管理器”面板中右击“数据连接”,在弹出的快捷菜单中选择“添加连接”命令,在弹出的“添加连接”对话框中,选择数据源为“Microsoft SQL Server 数据库文件 (SqlClient)”,单击“浏览”按钮,选择数据库文件名,这里是:E:\MicroBlog\BlogWebApp\App_Data\userlog.mdf。在连接数据库时,可以使用 SQL Server 身份验证,输入数据库用户名和密码,也可以使用 Windows 身份验证,这里使用 Windows 身份验证,单击“测试连接”按钮,测试是否连接成功。

9.1.4 在 Web. config 中部署数据库连接字符串

(1) 为了在 Web. config 中自动生成数据库连接字符串,可以创建一个测试用的 test. aspx Web 页面文件。

切换到“设计”视图。在页面上添加一个 SqlDataSource 数据源控件，单击 SqlDataSource 右上角的小三角按钮,在打开的“SqlDataSource 任务”对话框中,选择“配置数据源”选项。在打开的“配置数据源”对话框中,在“应用程序连接数据库应使用哪个数据连接?”下拉列表框中选择已有的连接 userlog. mdf。

可以单击“连接字符串”按钮,查看数据库连接串代码。

单击“下一步”按钮,在弹出的“配置数据源”对话框中,选择“是否将连接保存到应用配置文件中?”将连接字符串存储在应用配置文件中,可以简化维护和部署,否则连接字符串将作为数据源控件的属性保存在该页中。这里,选择“是”选项,并将此连接另存为 userlogConnectionString。

单击“下一步”按钮,在弹出的“配置数据源”对话框中,配置 Select 语句,选择指定来自表或视图的字段列,这里可以选择 article 表的所有字段列。

单击“下一步”按钮,在弹出的“配置数据源”对话框中,测试查询。预览从数据源返回的数据。最后单击“完成”按钮。

(2) 查看 Web. config 配置文件中的自动生成的数据库连接代码如下:

```
<?xml version = "1.0"?>
<configuration>
  <appSettings/>
    <connectionStrings>
      <add name = " userlogConnectionString " connectionString = " Data Source = .\SQLEXPRESS;AttachDbFilename = |DataDirectory|\userlog.mdf;Integrated Security = True;User Instance = True" providerName = "System.Data.SqlClient" />
    </connectionStrings>
...
  </configuration>
```

以上是使用 Windows 身份验证的数据库连接字符串。倘若使用 SQL Server 身份验证,要在数据库安全性设置中增加一个 SQL Server 账号,并将连接字符串中的 connectionString 属性值修改如下:

```
  < add name = " userlogConnectionString " connectionString = " Data Source = localhost\SQLEXPRESS;Initial Catalog = userlog;User ID = sa;password = 123456" />
```

其中,add 标记中的 name 属性用来指定连接字符串的名称;connectionString 属性用来存储连接字符串。在连接字符串参数中,Data Source 用来指定 SQL Server 数据库服务器的名称,Integrated Security 决定连接是否安全,Initial Catalog 用来指定数据库的名称。

在数据库访问时,可使用如下代码获取数据库连接字符串:

```
string connectionString =
ConfigurationManager.ConnectionStrings["userlogConnectionString"].ConnectionString;
```

Web. config 文件的完整结构与安全性配置将在下一个任务中继续讨论完成。

9.1.5　Blog 网站实体类的实现

在系统模型层 BlogModels 下编写实体类 User. cs、LogArticle. cs、Review. cs。下面分别列出 User 类、LogArticle 类和 Review 类的实现代码。

(1) User. cs 具体实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace BlogModels
{
    [Serializable()]
    public class User
    {
        private int user_id;
        private string user_name = String.Empty;
        private string user_pw = String.Empty;
        private string user_QQ = String.Empty;
        private DateTime login_date;
        public User() { }
        public int User_id
        {
            get { return this.user_id; }
            set { this.user_id = value; }
        }
        public string User_name
        {
            get { return this.user_name; }
            set { this.user_name = value; }
        }
        public string User_pw
        {
            get { return this.user_pw; }
            set { this.user_pw = value; }
        }
        public string User_QQ
        {
            get { return this.user_QQ; }
            set { this.user_QQ = value; }
        }
        public DateTime Login_date
        {
            get { return this.login_date; }
            set { this.login_date = value; }
        }
    }
}
```

（2）LogArticle.cs具体实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace BlogModels
{
    [Serializable()]
    public class LogArticle
    {
        private int arti_id;
        private int user_id;
        private string user_name = String.Empty;
        private string arti_title = String.Empty;
        private string arti_text = String.Empty;
        private int arti_click;
        private string arti_review = String.Empty;
        private DateTime submit_date;
        public LogArticle() { }

        public int Arti_id
        {
            get { return this.arti_id; }
            set { this.arti_id = value; }
        }
        public int User_id
        {
            get { return this.user_id; }
            set { this.user_id = value; }
        }
        public string User_name
        {
            get { return this.user_name; }
            set { this.user_name = value; }
        }
        public string Arti_title
        {
            get { return this.arti_title; }
            set { this.arti_title = value; }
        }
        public string Arti_text
        {
            get { return this.arti_text; }
            set { this.arti_text = value; }
        }
        public int Arti_click
        {
            get { return this.arti_click; }
            set { this.arti_click = value; }
```

```
        }
        public string Arti_review
        {
            get { return this.arti_review; }
            set { this.arti_review = value; }
        }
        public DateTime Submit_date
        {
            get { return this.submit_date; }
            set { this.submit_date = value; }
        }
    }
}
```

（3）Review.cs 具体实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace BlogModels
{
    [Serializable()]
    public class Review
    {
        private int revi_id;
        private int arti_id;
        private int user_id;
        private string revi_text = String.Empty;
        private string user_name = String.Empty;
        private string arti_title = String.Empty;
        private DateTime submit_date;
        public Review() { }

        public int Revi_id
        {
            get { return this.revi_id; }
            set { this.revi_id = value; }
        }
        public int Arti_id
        {
            get { return this.arti_id; }
            set { this.arti_id = value; }
        }
        public int User_id
        {
            get { return this.user_id; }
            set { this.user_id = value; }
        }
        public string    User_name
        {
```

```
            get { return this.user_name; }
            set { this.user_name = value; }
        }
        public string Arti_title
        {
            get { return this.arti_title; }
            set { this.arti_title = value; }
        }
        public string Revi_text
        {
            get { return this.revi_text; }
            set { this.revi_text = value; }
        }
        public DateTime Submit_date
        {
            get { return this.submit_date; }
            set { this.submit_date = value; }
        }
    }
}
```

9.1.6 Blog网站数据访问层的实现

在BlogDAL项目中编写与各实体类相对应的数据访问类：UserService类、ArticleService类和ReviewService类，以实现对各数据表的增、删、改、查。

(1) UserService.cs具体实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using BlogModels;
namespace BlogDAL
{
    public static class UserService
    {
        //从Web.Config中获取数据库连接字符串
        private static string strConn = System.Configuration.ConfigurationManager.
ConnectionStrings["userlogConnectionString"].ConnectionString;
        /// <summary>
        /// 查询用户是否存在
        /// </summary>

        public static User QueryUser(User inUser)
        {
            //根据用户输入的用户名和密码进行查找
            string strSql = string.Format("Select * From userinfo Where user_name='{0}'
```

```
And user_pw = '{1}'", inUser.User_name, inUser.User_pw);
            //创建 Connection 对象，使用 using 语句可以及时释放资源
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();    //打开数据库连接
                //初始化 System.Data.SqlClient.SqlCommand 类的新实例
                SqlCommand cmd = new SqlCommand(strSql, conn);
                //执行查询返回记录集
                SqlDataReader dr = cmd.ExecuteReader();
                if (dr.Read())
                {
                    User validUser = new User();
                    validUser.User_id = (int)dr["user_id"];
                    validUser.User_name = (string)dr["user_name"];
                    validUser.User_pw = (string)dr["user_pw"];
                    dr.Close();
                    return validUser;
                }
                else
                {
                    dr.Close();
                    return null;
                }
            }
        }

        /// <summary>
        /// 查询用户名是否已被使用
        /// </summary>
        public static bool QueryUserName(User inUser)
        {
            //根据用户输入的用户名查询
            string strSql = string.Format("Select * From userinfo Where user_name = '{0}'",
inUser.User_name);
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand(strSql, conn);
                SqlDataReader dr = cmd.ExecuteReader();
                if (dr.Read())
                {
                    dr.Close();
                    return true;    //用户名已被使用
                }
                else
                {
                    dr.Close();
                    return false;
                }
```

```
        }
    }

    /// <summary>
    /// 添加新用户
    /// </summary>
    public static void AddUser(User inUser)
    {
        string strSql = @"INSERT Userinfo(user_name,user_pw,QQ,login_date)" +
                        "VALUES (@user_name,@user_pw,@QQ,@login_date)";
        using (SqlConnection conn = new SqlConnection(strConn))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand(strSql, conn);
            cmd.Parameters.AddWithValue("@user_name", inUser.User_name);
            cmd.Parameters.AddWithValue("@user_pw", inUser.User_pw);
            cmd.Parameters.AddWithValue("@QQ", inUser.User_QQ);
            cmd.Parameters.AddWithValue("@login_date", System.DateTime.Now);
            cmd.ExecuteNonQuery();
        }
    }

    /// <summary>
    /// 根据 UserId 返回一个用户(User 对象)
    /// </summary>
    public static User GetUserByUserId(int userid)
    {
        string strSql = "SELECT * FROM userinfo WHERE user_id = @UserId";
        using (SqlConnection conn = new SqlConnection(strConn))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = conn;
            cmd.CommandText = strSql;
            cmd.Parameters.AddWithValue("@UserId", userid);
            SqlDataReader dr = cmd.ExecuteReader();
            if (dr.Read())
            {
                User user = new User();
                user.User_id = (int)dr["user_id"];
                user.User_name = (string)dr["user_name"];
                dr.Close();
                return user;
            }
            else
            {
                dr.Close();
                return null;
            }
```

```
            }
        }

        /// <summary>
        /// 返回全部用户
        /// </summary>
        public static IList<User> GetUserAll()
        {
            string strSql = "Select * From userinfo order by login_date DESC";
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = conn;
                cmd.CommandText = strSql;
                SqlDataReader dr = cmd.ExecuteReader();
                //使用 IList<T>传递实体对象集合
                List<User> list = new List<User>();
                while (dr.Read())
                {
                    User user = new User();
                    for (int i = 0; i < dr.FieldCount; i++)
                    {
                        string fieldName = dr.GetName(i);
                        if (fieldName == "user_id")
                            user.User_id = (int)dr["user_id"];
                        if (fieldName == "user_name")
                            user.User_name = (string)dr["user_name"];
                        if (fieldName == "user_pw")
                            user.User_pw = (string)dr["user_pw"];
                        if (fieldName == "QQ")
                            user.User_QQ = (string)dr["QQ"];
                        if (fieldName == "login_date")
                            user.Login_date = (DateTime)dr["login_date"];
                    }
                    list.Add(user);
                }
                dr.Close();
                return list;
            }
        }
    }
}
```

(2) ArticleService.cs 具体实现代码如下：

```
...
using BlogModels;
namespace BlogDAL
{
```

```
    public static class ArticleService
    {
        private static string strConn = System.Configuration.ConfigurationManager.
ConnectionStrings["userlogConnectionString"].ConnectionString;
        /// <summary>
        /// 添加日志文章
        /// </summary>
        public static int AddArticle(LogArticle article, string userid)
        {
            string strSql = @"INSERT article(user_id,arti_title,art i_text,arti_click,submit_
date)" + "VALUES (@user_id,@arti_title,@arti_text,@arti_click,@submit_date)";
             using (SqlConnection conn = new SqlConnection(strConn))
             {
                conn.Open();
                SqlCommand cmd = new SqlCommand(strSql, conn);
                //创建 System.Data.SqlClient.SqlCommand 的参数
                SqlParameter[] param = new SqlParameter[]
             {
             new SqlParameter("@user_id",SqlDbType.VarChar,10),
             new SqlParameter("@arti_title", SqlDbType.VarChar,50),
             new SqlParameter("@arti_text", SqlDbType.VarChar,100),
             new SqlParameter("@arti_click",SqlDbType.Int),
             new SqlParameter("@submit_date",SqlDbType.DateTime)
             };
                //给参数赋值
                param[0].Value = userid;
                param[1].Value = article.Arti_title;
                param[2].Value = article.Arti_text;
                param[3].Value = article.Arti_click;
                param[4].Value = System.DateTime.Now;
                //获取参数集合
                cmd.Parameters.AddRange(param);
                //执行 SQL 查询,返回受影响的行数
                return cmd.ExecuteNonQuery();
            }
        }

        /// <summary>
        /// 更新日志文章(ByArticleId——这里用来更新文章的阅读数)
        /// </summary>
        public static void UpdateArticle(LogArticle article)
        {
            string strSql = "UPDATE article SET arti_click = @arti_click WHERE arti_id = @
arti_id";
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = conn;
```

```
                cmd.CommandText = strSql;
                cmd.Parameters.AddWithValue("@arti_id", article.Arti_id);
                cmd.Parameters.AddWithValue("@arti_click", article.Arti_click);
                cmd.ExecuteNonQuery();
            }
        }

        /// <summary>
        /// 获取全部文章
        /// </summary>
        public static IList<LogArticle> GetArticleAll()
        {
            string strSql = "Select * From article order by submit_date DESC";
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = conn;
                cmd.CommandText = strSql;
                SqlDataReader dr = cmd.ExecuteReader();
                //使用 IList<T>传递实体对象集合
                List<LogArticle> list = new List<LogArticle>();

                while (dr.Read())
                {
                    LogArticle article = new LogArticle();
                    for (int i = 0; i < dr.FieldCount; i++ )
                    {
                        string fieldName = dr.GetName(i);
                        if (fieldName == "arti_id")
                            article.Arti_id = (int)dr["arti_id"];
                        if (fieldName == "user_id")
                        {
                            article.User_id = (int)dr["user_id"];
                            //使用外键对象返回用户名称
                            article.User_name = UserService.GetUserByUserId((int)dr
["user_id"]).User_name;
                        }
                        if (fieldName == "arti_title")
                            article.Arti_title = (string)dr["arti_title"];
                        if (fieldName == "arti_text")
                            article.Arti_text = (string)dr["arti_text"];
                        if (fieldName == "arti_click")
                            article.Arti_click = (int)dr["arti_click"];
                        if (fieldName == "submit_date")
                            article.Submit_date = (DateTime)dr["submit_date"];
                    }
                    list.Add(article);
                }
```

```
            dr.Close();
            return list;
        }
    }

    /// <summary>
    /// 返回一个用户的全部日志文章(ByUserId)
    /// </summary>
    public static IList<LogArticle> GetArticleByUserId(int userid)
    {
        string strSql = "SELECT * FROM article WHERE user_id = @UserId";
        using (SqlConnection conn = new SqlConnection(strConn))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = conn;
            cmd.CommandText = strSql;
            cmd.Parameters.AddWithValue("@UserId", userid);
            SqlDataReader dr = cmd.ExecuteReader();
            List<LogArticle> list = new List<LogArticle>();

            while (dr.Read())
            {
                LogArticle article = new LogArticle();
                for (int i = 0; i < dr.FieldCount; i++ )
                {
                    string fieldName = dr.GetName(i);
                    if (fieldName == "arti_id")
                        article.Arti_id = (int)dr["arti_id"];
                    if (fieldName == "user_id")
                    {
                        article.User_id = (int)dr["user_id"];
                        //使用外键对象返回用户名称
                         article.User_name = UserService.GetUserByUserId((int)dr
["user_id"]).User_name;
                    }
                    if (fieldName == "arti_title")
                        article.Arti_title = (string)dr["arti_title"];
                    if (fieldName == "arti_text")
                        article.Arti_text = (string)dr["arti_text"];
                    if (fieldName == "arti_click")
                        article.Arti_click = (int)dr["arti_click"];
                    if (fieldName == "submit_date")
                        article.Submit_date = (DateTime)dr["submit_date"];
                }
                list.Add(article);
            }
            dr.Close();
            return list;
```

```
        }
    }

    /// <summary>
    /// 根据 ArticleId 返回一篇文章(Article 对象)
    /// </summary>
    public static LogArticle GetArticleByArticleId(int articleid)
    {
        string strSql = "SELECT * FROM article WHERE arti_id = @articleid";
        using (SqlConnection conn = new SqlConnection(strConn))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = conn;
            cmd.CommandText = strSql;
            cmd.Parameters.AddWithValue("@articleid", articleid);
            SqlDataReader dr = cmd.ExecuteReader();
            if (dr.Read())
            {
                LogArticle article = new LogArticle();
                article.Arti_id = (int)dr["arti_id"];
                article.User_id = (int)dr["user_id"];
                //使用外键对象返回用户名称
                article.User_name = UserService.GetUserByUserId((int)dr["user_
id"]).User_name;
                article.Arti_title = (string)dr["arti_title"];
                article.Arti_text = (string)dr["arti_text"];
                article.Arti_click = (int)dr["arti_click"];
                article.Submit_date = (DateTime)dr["submit_date"];
                dr.Close();
                return article;
            }
            else
            {
                dr.Close();
                return null;
            }
        }
    }

    /// <summary>
    /// 删除文章
    /// </summary>
    public static void DeleteArticle(LogArticle article)
    {
        string strSql = "DELETE article WHERE arti_id = @arti_id";
        using (SqlConnection conn = new SqlConnection(strConn))
        {
            conn.Open();
```

```
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = conn;
                cmd.CommandText = strSql;
                cmd.Parameters.AddWithValue("@arti_id", article.Arti_id);
                cmd.ExecuteNonQuery();
            }
        }

        /// <summary>
        /// 获得文章总数
        /// </summary>
        public static int GetArticleNumber()
        {
            string strSql = "SELECT count(*) FROM article";
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = conn;
                cmd.CommandText = strSql;
                int article_num = Convert.ToInt32(cmd.ExecuteScalar());
                return article_num;
            }
        }

        /// <summary>
        /// 截取文章前面部分内容
        /// </summary>
        public static string GetPartString(string article_text, int n)
        {
            if (article_text.ToString().Length > n - 2)
                return article_text.ToString().Substring(0, n - 2) + "...";
            else
                return article_text.ToString();
        }
    }
}
```

(3) ReviewService.cs 具体实现代码如下：

```
...
using BlogModels;
namespace BlogDAL
{
    public static class ReviewService
    {
        private static string strConn = System.Configuration.ConfigurationManager.
ConnectionStrings["userlogConnectionString"].ConnectionString;
        /// <summary>
        /// 添加新评论
```

```
        /// </summary>
        public static int AddReview(Review review, int articleid)
        {
            string strSql = @"INSERT review(arti_id,user_id,revi_text,submit_date)" +
    "VALUES (@arti_id,@user_id,@revi_text,@submit_date)";
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand(strSql, conn);
                cmd.Parameters.AddWithValue("@arti_id", articleid);
                cmd.Parameters.AddWithValue("@user_id", review.User_id);
                cmd.Parameters.AddWithValue("@revi_text", review.Revi_text);
                cmd.Parameters.AddWithValue("@submit_date", System.DateTime.Now);
                return cmd.ExecuteNonQuery();
            }
        }

        /// <summary>
        /// 获取某篇文章的全部评论(ByArticleId)
        /// </summary>
        public static IList<Review> GetReviewByArticleId(int articleid)
        {
            string strSql = "SELECT * FROM review WHERE arti_id = @ArticleId";
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = conn;
                cmd.CommandText = strSql;
                cmd.Parameters.AddWithValue("@ArticleId", articleid);
                SqlDataReader dr = cmd.ExecuteReader();
                List<Review> list = new List<Review>();
                while (dr.Read())
                {
                    Review review = new Review();
                    for (int i = 0; i < dr.FieldCount; i++)
                    {
                        string fieldName = dr.GetName(i);
                        if (fieldName == "revi_id")
                            review.Revi_id = (int)dr["revi_id"];
                        if (fieldName == "arti_id")
                            review.Arti_id = (int)dr["arti_id"];
                        //使用外键对象返回文章标题
                        review.Arti_title = ArticleService.GetArticleByArticleId((int)
dr["arti_id"]).Arti_title;
                        if (fieldName == "user_id")
                        {
                            review.User_id = (int)dr["user_id"];
                            //使用外键对象返回用户名称
```

```
                            review.User_name = UserService.GetUserByUserId((int)dr
["user_id"]).User_name;
                        }
                        if (fieldName == "revi_text")
                            review.Revi_text = (string)dr["revi_text"];
                        if (fieldName == "submit_date")
                            review.Submit_date = (DateTime)dr["submit_date"];
                    }
                    list.Add(review);
                }
                dr.Close();
                return list;
            }
        }

        /// <summary>
        /// 获得某篇文章的评论数(By articleid)
        /// </summary>
        public static int GetReviewNumber(int articleid)
        {
            string strSql = "SELECT count(*) FROM review WHERE arti_id = @articleid";
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = conn;
                cmd.CommandText = strSql;
                cmd.Parameters.AddWithValue("@articleid", articleid);
                int review_num = Convert.ToInt32(cmd.ExecuteScalar());
                return review_num;
            }
        }

        /// <summary>
        /// 返回最新若干评论
        /// </summary>
        public static IList<Review> GetReviewNew()
        {
            string strSql = "Select top 4 * From review order by submit_date DESC";
            using (SqlConnection conn = new SqlConnection(strConn))
            {
                conn.Open();
                SqlCommand cmd = new SqlCommand();
                cmd.Connection = conn;
                cmd.CommandText = strSql;
                SqlDataReader dr = cmd.ExecuteReader();
                List<Review> list = new List<Review>();
                while (dr.Read())
                {
```

```
                Review review = new Review();
                for (int i = 0; i < dr.FieldCount; i++ )
                {
                    string fieldName = dr.GetName(i);
                    if (fieldName == "revi_id")
                        review.Revi_id = (int)dr["revi_id"];
                    if (fieldName == "user_id")
                    {
                        review.User_id = (int)dr["user_id"];
                        //使用外键对象返回用户名称
                        review.User_name = UserService.GetUserByUserId((int)dr
["user_id"]).User_name;
                    }
                    if (fieldName == "arti_id")
                    {
                        review.Arti_id = (int)dr["arti_id"];
                        //使用外键对象返回文章标题
                        review.Arti_title = ArticleService.GetArticleByArticleId
((int)dr["arti_id"]).Arti_title;
                    }
                    if (fieldName == "revi_text")
                        review.Revi_text = (string)dr["revi_text"];
                    if (fieldName == "submit_date")
                        review.Submit_date = (DateTime)dr["submit_date"];
                }
                list.Add(review);
            }
            dr.Close();
            return list;
        }
    }

    /// <summary>
    /// 返回所有评论
    /// </summary>
    public static IList<Review> GetReviewAll()
    {
        string strSql = "Select * From review order by submit_date DESC";
        using (SqlConnection conn = new SqlConnection(strConn))
        {
            conn.Open();
            SqlCommand cmd = new SqlCommand();
            cmd.Connection = conn;
            cmd.CommandText = strSql;
            SqlDataReader dr = cmd.ExecuteReader();
            List<Review> list = new List<Review>();
            while (dr.Read())
            {
                Review review = new Review();
```

```
                        for (int i = 0; i < dr.FieldCount; i++ )
                        {
                            string fieldName = dr.GetName(i);
                            if (fieldName == "revi_id")
                                review.Revi_id = (int)dr["revi_id"];
                            if (fieldName == "user_id")
                            {
                                review.User_id = (int)dr["user_id"];
                                //使用外键对象返回用户名称
                                review.User_name = UserService.GetUserByUserId((int)dr
["user_id"]).User_name;
                            }
                            if (fieldName == "arti_id")
                            {
                                review.Arti_id = (int)dr["arti_id"];
                                //使用外键对象返回文章标题
                                review.Arti_title = ArticleService.GetArticleByArticleId
((int)dr["arti_id"]).Arti_title;
                            }
                            if (fieldName == "revi_text")
                                review.Revi_text = (string)dr["revi_text"];
                            if (fieldName == "submit_date")
                                review.Submit_date = (DateTime)dr["submit_date"];
                        }
                        list.Add(review);
                    }
                    dr.Close();
                    return list;
                }
            }
        }
    }
```

9.1.7 Blog 网站业务逻辑层的实现

业务逻辑层负责数据访问层与表示层之间的数据传递和业务逻辑处理。在 BlogBLL 项目中编写与数据访问层相对应的类和方法如下。

（1）UserManager.cs 具体实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
using BlogModels;
using BlogDAL;
namespace BlogBLL
{
    public static class UserManager
    {
        public static bool UserLogin(User inUser,out User userExist)
```

```
        {
            User validUser = UserService.QueryUser(inUser);
            if (validUser == null)
            {
                userExist = null;    //用户登录名密码不正确
                return false;
            }
            else
            {
                userExist = validUser;
                return true;
            }
        }
        /// <summary>
        /// 新用户注册、先判断用户名是否已被使用
        /// </summary>
        public static bool UserRegister(User inUser)
        {
            if (UserService.QueryUserName(inUser))
            {
                return false;    //用户名已被使用
            }
            else
            {
                UserService.AddUser(inUser);
                return true;
            }
        }
        /// <summary>
        /// 根据 UserId 查询返回用户名(User 对象)
        /// </summary>
        public static User GetUserByUserId(int userid)
        {
            return UserService.GetUserByUserId(userid);
        }
        /// <summary>
        /// 返回所有用户
        /// </summary>
        public static IList<User> GetUserALL()
        {
            return UserService.GetUserAll();
        }
    }
}
```

（2）ArticleManager.cs 具体实现代码如下：

```
…
using BlogModels;
using BlogDAL;
```

```
namespace BlogBLL
{
    public static class ArticleManager
    {
        /// <summary>
        /// 添加新文章
        /// </summary>
        public static int  AddArticle(LogArticle article, string  usrid)
        {
            return ArticleService.AddArticle(article,usrid);
        }
        /// <summary>
        /// 返回所有文章
        /// </summary>
        public static IList<LogArticle> GetArticleAll()
        {
            return ArticleService.GetArticleAll();
        }
        /// <summary>
        /// 获得文章总数
        /// </summary>
        public static int GetArticleNumber()
        {
            return ArticleService.GetArticleNumber();
        }
        /// <summary>
        /// 返回一个用户的全部日志文章(ByUserId)
        /// </summary>
        public static IList<LogArticle> GetArticleByUserId(int userid)
        {
            return ArticleService.GetArticleByUserId(userid);
        }
        /// <summary>
        /// 根据 ArticleId 返回一篇文章
        /// </summary>
        public static LogArticle GetArticleByArticleId(int articleid)
        {
            return ArticleService.GetArticleByArticleId(articleid);
        }
        /// <summary>
        /// 修改文章
        /// </summary>
        public static void UpdateArticle(LogArticle article)
        {
            ArticleService.UpdateArticle(article);
        }
        /// <summary>
        /// 删除文章
        /// </summary>
```

```
        public static void DeleteArticle(LogArticle article)
        {
            ArticleService.DeleteArticle(article);
        }
        /// < summary>
        /// 截取文章的前面部分内容
        /// </summary>
        public static string GetPartString(string article_text, int num)
        {
            return ArticleService.GetPartString(article_text, num);
        }
    }
}
```

（3）ReviewManager.cs 具体实现代码如下：

```
...
using BlogModels;
using BlogDAL;
namespace BlogBLL
{
    public static class ReviewManager
    {
        /// < summary>
        /// 添加新评论
        /// </summary>
         public static int AddReview(Review review,int articleid)
        {
            return  ReviewService.AddReview(review,articleid);
        }
        /// < summary>
        /// 返回最新评论
        /// </summary>
        public static IList<Review> GetReviewNew()
        {
            return ReviewService.GetReviewNew();
        }
        /// < summary>
        /// 返回所有评论
        /// </summary>
        public static IList<Review> GetReviewAll()
        {
            return ReviewService.GetReviewAll();
        }
        /// < summary>
        /// 返回一篇文章的全部评论(Byarticleid)
        /// </summary>
        public static IList<Review> GetReviewByArticleId(int articleid)
        {
            return  ReviewService.GetReviewByArticleId(articleid);
```

```
            }
            /// <summary>
            /// 返回一篇文章的评论总数(Byarticleid)
            /// </summary>
            public static int GetReviewNumber(int articleid)
            {
                return ReviewService.GetReviewNumber(articleid);
            }
        }
    }
```

9.1.8 Web表示层的实现

表示层就是整个Web站点，它直接与用户交互。表示层页面的主要设计是控件＋事件：将控件绑定数据，并在需要时编写事件代码调用业务逻辑层的方法。

（1）日志网站首页DefaultArticleAllRepeat.aspx运行效果如图9-2所示。实现代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" MasterPageFile="~/MicroBlog.Master"
  Codebehind="DefaultArticleAllRepeat.aspx.cs" Inherits="BlogWebApp.DefaultArticleAllRepeat" %>

<%@ Register Assembly="AspNetPager" Namespace="Wuqi.Webdiyer" TagPrefix="webdiyer" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
     <div class="ArticleAll_layer">
        <div id="text">全部日志文章</div>
        <asp:Repeater ID="Repeater1" runat="server">
            <ItemTemplate>
                <asp:Label ID="Label1" runat="server" Text='<%# Eval("Arti_id") %>'
Visible="false"></asp:Label><br />
                <asp:Label ID="Label2" runat="server" Text='<%# Eval("User_id") %>'
Visible="false"></asp:Label><br />
                UserName:<%# Eval("User_name") %>    发表日期:<%#
Eval("Submit_date") %><br />
                标题:<%# Eval("Arti_title") %><br />
                内容:<%# GetPartString(Eval("Arti_text").ToString(),180) %><br />
                阅读(<%# Eval("Arti_click") %>)    <a href =
"ReviewByArticleId.aspx?articleid=<%# Eval("Arti_id") %>">
                评论(<%# GetReviewNumber(Convert.ToInt32(Eval("Arti_id"))) %>)</a>
<br />
                <a href="ArticleDetailByArticleId.aspx?articleid=<%# Eval("Arti_
id") %>">查看全文</a>
             </ItemTemplate>
        </asp:Repeater><br />
         <webdiyer:AspNetPager ID="AspNetPager1" runat="server" UrlPaging="true"
PageSize="2"
              ShowPageIndex="true" AlwaysShow="true" Width="317px" OnPageChanged=
"AspNetPager1_PageChanged">
        </webdiyer:AspNetPager>
     </div>
```

```
</asp:Content>

<asp:Content ID = "Content2" ContentPlaceHolderID = "ContentPlaceHolder2" runat = "server">
    <div class = "Login_layer">
        <table>
            <tr><td>
                    <asp:Label ID = "Label3" runat = "server" Text = "用户名"></asp:Label>
                    <asp:TextBox ID = "txt_name" runat = "server"></asp:TextBox></td></tr>
            <tr><td>
                    <asp:Label ID = "Label4" runat = "server" Text = "密码"></asp:Label>
                     <asp:TextBox ID = "txt_pw" runat = "server" TextMode = "Password">
</asp:TextBox></td></tr>
            <tr><td>
                    <asp:Button ID = "btn_login" runat = "server" Text = "登录" OnClick =
"btn_login_Click" />
                    <asp:LinkButton ID = "LinkButton1" runat = "server" Text = "新用户注
册" PostBackUrl = "~/Login/NewLogin.aspx"></asp:LinkButton></td></tr>
        </table>
     </div><br />
</asp:Content>

<asp:Content ID = "Content3" ContentPlaceHolderID = "ContentPlaceHolder3" runat = "server">
    <div class = " GetUserALL_layer">
        <div>
            <asp:Label ID = "txt_msg" runat = "server" Text = "用户列表"></asp:Label></div>
        <asp:DataList ID = "DataList1" runat = "server" DataSourceID = "ObjectDataSource1">
            <ItemTemplate>
                <a href = "ArticleAllByUserId.aspx?userid = <% # Eval("User_id") %>"
target = "_blank"><% # Eval("User_name") %></a>
            </ItemTemplate>
        </asp:DataList>
        <asp:ObjectDataSource ID = "ObjectDataSource1" runat = "server" SelectMethod =
"GetUserALL"
            TypeName = "BlogBLL.UserManager">
        </asp:ObjectDataSource>
    </div>
</asp:Content>

<asp:Content ID = "Content4" ContentPlaceHolderID = "ContentPlaceHolder4" runat = "server">
    <div class = "GetReviewNew_layer">
        <asp:Label ID = "Label5" runat = "server" Text = "最新评论"></asp:Label><br />
        <asp:DataList ID = "DataList2" runat = "server" DataSourceID = "ObjectDataSource2">
            <ItemTemplate>
                <a href = "ArticleDetailByArticleId.aspx?articleid = <% # Eval("Arti_
id") %>" target = "_blank">«<% # Eval("Arti_title") %>»</a></br>
                <% # Eval("User_name") %>
                : <% # Eval("Submit_date") %></br>
                <% # Eval("Revi_text") %></br>
                <div><a href = "" target = "">回复评论</a></div></br>
```

```
            </ItemTemplate>
        </asp:DataList>
        <asp:ObjectDataSource ID = "ObjectDataSource2" runat = "server" SelectMethod =
"GetReviewNew" TypeName = "BlogBLL.ReviewManager">
        </asp:ObjectDataSource>
    </div>
    <div>
        <a href = "ReviewAllList.aspx" target = "_blank">全部评论>></a>
    </div>
</asp:Content>
```

在 DefaultArticleAllRepeat.aspx.cs 中编写如下代码：

```
...
using BlogModels;
using BlogBLL;
namespace BlogWebApp
{
    public partial class DefaultArticleAllRepeat : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)    //首次加载
            {
                //调用 PageBind()方法,为 Repeater 控件实现分页功能
                PageBind();
            }
        }

        // 截取文章的前面部分内容
        public static string GetPartString(string article_text, int num)
        {
            return ArticleManager.GetPartString(article_text, num);
        }

        // 根据 ArticleId 获得某篇文章的评论总数
        public static int GetReviewNumber(int articleid)
        {
            return ReviewManager.GetReviewNumber(articleid);
        }
        //使用 PagedDataSource 给 Repeater 控件增加分页功能
        private void PageBind()
        {
            //创建 PagedDataSource 对象实例
            PagedDataSource PageDataSource1 = new PagedDataSource();
            //设置 PagedDataSource 对象的属性值
            PageDataSource1.DataSource = ArticleManager.GetArticleAll();
            PageDataSource1.AllowPaging = true;
            PageDataSource1.PageSize = AspNetPager1.PageSize;
            PageDataSource1.CurrentPageIndex = AspNetPager1.CurrentPageIndex - 1;
```

```
            //设置分页控件的 RecordCount
            AspNetPager1.RecordCount = ArticleManager.GetArticleNumber();
            //绑定数据源
            Repeater1.DataSource = PageDataSource1;
            Repeater1.DataBind();
        }

        protected void AspNetPager1_PageChanged(object sender, EventArgs e)
        {
            PageBind();
        }

        protected void btn_login_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                User user = new User();
                user.User_name = txt_name.Text.Trim();
                user.User_pw = txt_pw.Text.Trim();
                //判断用户名和密码是否正确
                if (UserManager.UserLogin(user, out user))
                {
                    //保存用户主键标识
                    Session["user_id"] = user.User_id;
                    //重定向到发表日志页面
                    Response.Redirect("~/IssueArticleFckeditor.aspx");
                }
                else
                {
                    txt_msg.Text = "请查证并输入有效的用户名和密码!";
                    txt_name.Text = string.Empty;
                    txt_pw.Text = string.Empty;
                }
            }
        }
    }
}
```

说明：

① 在首页放置了与母版页(MicroBlog.Master)的内容位置控件相对应的 4 个内容控件(Content1～Content4)，分别用来显示全部日志文章、用户登录、所有用户列表、最新评论等模块。

② 这里用 Repeater 控件来实现全部日志文章列表。由于 Repeater 控件没有自带分页功能，因此使用了分页控件 AspNetPager。这是一个第三方控件，可以参照本书第 4 章中第三方控件的使用方法，下载 AspNetPager.dll、“添加引用”至 Web 层的 Bin 文件夹下，并且在工具箱中添加 AspNetPager 控件。

注意： AspNetPager 分页控件数据源绑定的方法，这里使用了 PagedDataSource 对

象。在 DefaultArticleAllRepeat.aspx.cs 的 PageBind()方法中给出了对 PagedDataSource 对象属性值的设置。

分页控件的 AspNetPager1_PageChanged()事件在页码改变时被激活。

数据分页是 Web 应用程序数据访问时的常用的功能之一。迄今为止,ASP.NET 的 DataList 控件、Repeater 控件等没有自带分页功能,分页代码的编程技术难度较大而且代码重用率较低。

AspNetPager 分页控件是一款优秀的第三方控件,它的分页数据可以来自任何数据源,例如 SQL Server、Oracle、mysql 等数据库以及 XML 文件等,它的控件和数据显示可以是相互独立的。AspNetPager 控件可以在它的官方网站 http://www.webdiyer.com/download 中下载。它的其他功能及属性设置,比如分页按钮的样式设置等,可登录它的网站查阅更多的详细说明。

③ GetPartString(Eval("Arti_text").ToString(),180)方法用来截取文章的前面部分字符串。

GetReviewNumber(Convert.ToInt32(Eval("Arti_id")))用来获取一篇文章的评论总数。

在数据访问层的 ArticleService.cs 类和 ReviewService.cs 类中分别给出了 GetPartString()方法和 GetReviewNumber()方法的定义。

④ 在 Login.aspx 页面文件中,用户提交的登录信息经过数据库验证后倘若证实有效,则要将用户主键标识 User_id 保存在 Session["user_id"] 中。在发表日志页面时,需要先判断 Session["user_id"] 是否为 null。

(2) 在首页的"用户登录"区域中可以选择新用户注册。NewLogin.aspx 新用户注册页面主要代码如下:

```
<%@ Page Language="C#" AutoEventWireup="true" MasterPageFile="~/MicroBlog.Master"
    Codebehind="NewLogin.aspx.cs" Inherits="BlogWebApp.Login.NewLogin" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div>
        <table>
            <tr><td><asp:Label ID="Label1" runat="server" Text="用户名"></asp:
Label></td>
                <td><asp:TextBox ID="txt_name" runat="server" Width="150px"></asp:
TextBox>
                    <asp:RequiredFieldValidator ID="rfv_name" runat="server"
ErrorMessage="请输入用户名" ControlToValidate="txt_name"></asp:RequiredFieldValidator>
</td></tr>
            <tr><td><asp:Label ID="Label2" runat="server" Text="密码" Width=
"48px"></asp:Label></td>
                <td><asp:TextBox ID="txt_pw" runat="server" TextMode="Password"
Width="150px"></asp:TextBox>
                    <asp:RequiredFieldValidator ControlToValidate="txt_pw" ID="rfv_
pw" runat="server" ErrorMessage="请输入密码"></asp:RequiredFieldValidator></td></tr>
            <tr><td><asp:Label ID="Label3" runat="server" Text="QQ" Width="49px">
```

```
</asp:Label></td>
                    <td><asp:TextBox ID = "txt_QQ" runat = "server" Width = "150px"></asp:
TextBox>
                        <asp:RequiredFieldValidator ID = "rfv_QQ" runat = "server" ErrorMessage =
"请输入 QQ 号" ControlToValidate = "txt_QQ"></asp:RequiredFieldValidator></td></tr>
                <tr><td colspan = "2"><asp:Button ID = "Button1" runat = "server" Text = "注
册" OnClick = "Button1_Click" /></td></tr>
            </table>
        </div>
</asp:Content>
```

NewLogin.aspx.cs 主要代码如下：

```
…
using BlogModels;
using BlogBLL;
namespace BlogWebApp.Login
{
    public partial class NewLogin : System.Web.UI.Page
    {
        protected void Button1_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                User user = new User();
                user.User_name = txt_name.Text.Trim();
                user.User_pw = txt_pw.Text.Trim();
                user.User_QQ = txt_QQ.Text.Trim();
                if (!UserManager.UserRegister(user))
                {
                    Response.Write("<script>alert('sorry,用户名已经存在!')</script>");
                }
                else
                {
                    Response.Write("<script>alert('注册成功!')</script>");
                }
            }
        }
    }
}
```

（3）在首页的“用户列表”中单击某个用户名，将显示这个用户的所有文章。显示一个用户所有文章的 ArticleAllByUserId.aspx 页面实现代码如下：

```
<%@ Page Language = "C#" AutoEventWireup = "true" Codebehind = "ArticleAllByUserId.aspx.cs"
    MasterPageFile = "~/MicroBlog.Master" Inherits = "BlogWebApp.ArticleAllByUserId" %>

<asp:Content ID = "Content1" ContentPlaceHolderID = "ContentPlaceHolder1" runat = "server">
    <div>
        <asp:DataList ID = "DataList1" runat = "server" DataSourceID = "ObjectDataSource1">
```

```
                <ItemTemplate>
                    <asp:Label ID="Label1" runat="server" Text='<%# Eval("Arti_id") %>'
Visible="false"></asp:Label><br />
                    <asp:Label ID="Arti_idLabel" runat="server" Text='<%# Eval("User_
id") %>' Visible="false"></asp:Label><br />
                    UserName:<%# Eval("User_name") %><br />
                    标题:<%# Eval("Arti_title") %><br />
                    内容:<%# GetPartString(Eval("Arti_text").ToString(),50)%><br />
                    发表日期:<%# Eval("Submit_date") %><br />
                    阅读(<%# Eval("Arti_click") %>)    <a href =
"ReviewByArticleId.aspx?articleid=<%# Eval("Arti_id") %>">
                        评论(<%# GetReviewNumber(Convert.ToInt32(Eval("Arti_id")))%>)
</a><br />
                    <a href="ArticleDetailByArticleId.aspx?articleid=<%# Eval("Arti_
id") %>">查看全文</a></p>
                </ItemTemplate>
            </asp:DataList>
            <asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod =
"GetArticleByUserId"
                TypeName="BlogBLL.ArticleManager">
                <SelectParameters>
                    <asp:QueryStringParameter Name="userid" QueryStringField="userid"
Type="Int32" />
                </SelectParameters>
            </asp:ObjectDataSource>
        </div>
</asp:Content>
```

在 ArticleAllByUserId.aspx.cs 中编写代码如下：

```
...
using BlogModels;
using BlogBLL;
namespace BlogWebApp
{
    public partial class ArticleAllByUserId : System.Web.UI.Page
    {
        /// <summary>
        /// 截取文章的前面部分内容
        /// </summary>
        public static string GetPartString(string article_text, int num)
        {
            return  ArticleManager.GetPartString(article_text, num);
        }

        /// <summary>
        /// 根据 ArticleId 获得某篇文章的评论总数
        /// </summary>
        public static int GetReviewNumber(int articleid)
        {
```

```
                return ReviewManager.GetReviewNumber(articleid);
            }
        }
    }
```

(4) 查看日志全文时，可在文章下方提交发表评论。ArticleDetailByArticleId.aspx "查看全文"页面实现代码如下，运行效果如图 9-4 所示。

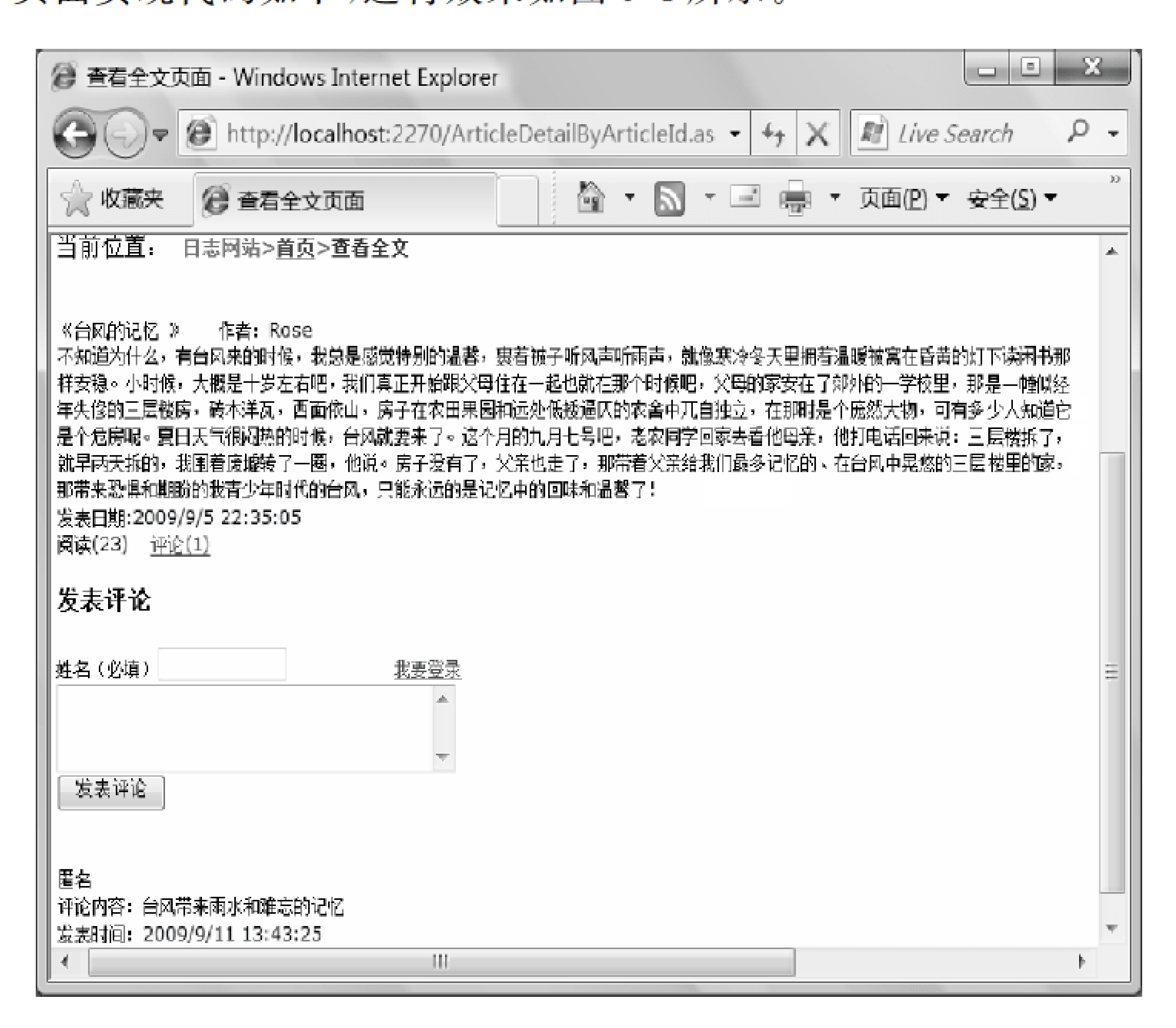


图 9-4　"查看全文"页面运行效果

```
<%@ Page Language="C#" AutoEventWireup="true" MasterPageFile="~/MicroBlog1.Master"
    Codebehind=" ArticleDetailByArticleId. aspx. cs " Inherits = " BlogWebApp.
ArticleDetailByAriticleId" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div class="ArticleDetail">
        <asp:ObjectDataSource ID = " ObjectDataSource1" runat = " server" SelectMethod =
"GetArticleByArticleId"
            TypeName="BlogBLL.ArticleManager">
            <SelectParameters>
                <asp: QueryStringParameter Name = " articleid " QueryStringField =
"articleid" Type="Int32" />
            </SelectParameters>
        </asp:ObjectDataSource>
        <asp:DataList ID="DataList1" runat="server" DataSourceID="ObjectDataSource1">
            <ItemTemplate>
                <asp:Label ID="Label1" runat="server" Text = '<%# Eval("Arti_id") %>'
Visible="false"></asp:Label><br />
```

```
                <asp:Label ID = "Arti_idLabel" runat = "server" Text = '<% # Eval("User_
id") %>' Visible = "false"></asp:Label><br />
                «<% # Eval("Arti_title") %>»      作者:
<% # Eval("User_name") %><br />
                <% # Eval("Arti_text") %><br />
                发表日期:<% # Eval("Submit_date") %><br />
                阅读(<% # Eval("Arti_click") %>)    <a href =
"ReviewByArticleId.aspx?articleid = <% # Eval("Arti_id") %>">
                    评论(<% # GetReviewNumber(Convert.ToInt32(Eval("Arti_id"))) %>)</a>
            </ItemTemplate>
        </asp:DataList>
    </div>
    <div class = "IssueReview.aspx">
        <h4>发表评论</h4>
        姓名(必填)<asp:TextBox ID = "txt_name" runat = "server" Width = "86px"></asp:
TextBox>
        <asp:RequiredFieldValidator ID = "RequiredFieldValidator1" ControlToValidate =
"txt_name" runat = "server" ErrorMessage = "请填写姓名!"></asp:RequiredFieldValidator>
<a href = "">我要登录</a><br />
        <asp:TextBox ID = "txt_text" runat = "server" Height = "50px" Width = "278px"
TextMode = "MultiLine"></asp:TextBox> <br />
        <asp:Button ID = "Button1" runat = "server" Text = "发表评论" OnClick = "Button1_Click" />
    </div>
    <div class = "ReviewList">
        <asp:ObjectDataSource ID = "ObjectDataSource2" runat = "server" SelectMethod =
"GetReviewByArticleId" TypeName = "BlogBLL.ReviewManager">
            <SelectParameters>
                <asp: QueryStringParameter Name = "articleid" QueryStringField =
"articleid" Type = "Int32" />
            </SelectParameters>
        </asp:ObjectDataSource>
        <asp:DataList ID = "DataList2" runat = "server" DataSourceID = "ObjectDataSource2">
            <ItemTemplate>
                <asp:Label ID = "Revi_id" runat = "server" Text = '<% # Eval("Revi_id") %>'
Visible = "false"></asp:Label><br />
                <asp:Label ID = "Arti_id" runat = "server" Text = '<% # Eval("Arti_id") %>'
Visible = "false"></asp:Label><br />
                <% # Eval("User_name") %><br />
                评论内容: <% # Eval("Revi_text") %><br />
                发表时间: <% # Eval("Submit_date") %><br />
            </ItemTemplate>
        </asp:DataList>
    </div>
</asp:Content>
```

在 ArticleDetailByArticleId.aspx.cs 中编写如下代码:

```
...
using BlogModels;
using BlogBLL;
```

```
namespace BlogWebApp
{
    public partial class ArticleDetailByAriticleId : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                if (Request.QueryString["articleid"] != null)
                {
                    int articleid;
                    try
                    {
                        articleid = Convert.ToInt32(Request.QueryString["articleid"]);
                        LogArticle article = ArticleManager.GetArticleByArticleId(articleid);
                        //文章阅读次数加 1
                        article.Arti_click += 1;
                        ArticleManager.UpdateArticle(article);
                        this.DataList1.DataBind();
                    }
                    catch
                    {
                        Response.Write("<script>alert('Sorry,此文不存在!')</script>");
                    }
                }
            }
        }
        protected void Button1_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                //获取文章 ID
                int articleid = Convert.ToInt32(Request.QueryString["articleid"]);
                Review review = new Review();
                //获取评论内容
                review.Revi_text = txt_text.Text.Trim().ToString();
                review.Submit_date = DateTime.Now; //发表评论时间
                if (Session["user_id"] == null)
                {
                    //给匿名评论用户设定 User_id
                    review.User_id = 80;
                    review.User_name = txt_name.Text.Trim().ToString();
                }
                else
                {
                    int userid = Convert.ToInt32((Session["user_id"]).ToString());
                    //获取已登录评论用户的 User_id
                    review.User_id = userid;
                    review.User_name = UserManager.GetUserByUserId(userid).User_name;
```

```
                    }
                    if (ReviewManager.AddReview(review, articleid) != 0)
                    {
                        Response.Redirect("ArticleDetailByArticleId.aspx?articleid=" + articleid);
                        Response.Write("<script>alert('评论发表成功!')</script>");
                    }
                    else
                    {
                        Response.Write("<script>alert('Sorry,评论提交失败!')</script>");
                    }
                }
            }
            /// <summary>
            /// 根据 ArticleId 获得某篇文章的评论总数
            /// </summary>
            public static int GetReviewNumber(int articleid)
            {
                return ReviewManager.GetReviewNumber(articleid);
            }
        }
    }
```

(5) 在首页“全部日志文章”中或是在“查看全文”页面中单击“评论”超链接，可以看到一篇文章的全部评论。显示一篇文章的全部评论的 ReviewByArticleId.aspx 页面实现代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" MasterPageFile="~/MicroBlog1.Master"
    Codebehind="ReviewByArticleId.aspx.cs" Inherits="BlogWebApp.ReviewByArticleId" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div class=" Review_layer ">
        <asp:DataList ID="DataList1" runat="server" DataSourceID="ObjectDataSource1">
            <ItemTemplate>
                <asp:Label ID="Label1" runat="server" Text='<%# Eval("Revi_id") %>'
Visible="false"></asp:Label><br />
                <asp:Label ID="Label2" runat="server" Text='<%# Eval("Arti_id") %>'
Visible="false"></asp:Label><br />
                «<%# Eval("Arti_title") %>»<%# Eval("User_name") %><br />
                评论内容:<%# Eval("Revi_text") %><br />
                发表时间:<%# Eval("Submit_date")%><br />
            </ItemTemplate>
        </asp:DataList>
        <asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod=
"GetReviewByArticleId"
            TypeName="BlogBLL.ReviewManager">
            <SelectParameters>
                <asp:QueryStringParameter Name="articleid" QueryStringField="articleid"
Type="Int32" />
            </SelectParameters>
```

```
        </asp:ObjectDataSource>
    </div>
</asp:Content>
```

(6) 在首页左侧“最新评论”区域下方单击“全部评论”超链接,可以看到所有文章的全部评论。ReviewAllList.aspx 页面实现代码如下:

```
<%@ Page Language="C#" AutoEventWireup="true" MasterPageFile="~/MicroBlog1.Master"
    Codebehind="ReviewAllList.aspx.cs" Inherits="BlogWebApp.ReviewAllList" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div class="ReviewAll_layer">
        <div id="text">全部评论</div>
        <asp:DataList ID="DataList1" runat="server" DataSourceID="ObjectDataSource1">
            <ItemTemplate>
                <a href="ArticleDetailByArticleId.aspx?articleid=<%# Eval("Arti_
id") %>" target="_blank">«<%# Eval("Arti_title") %>»</a></br>
                <%# Eval("User_name") %>
                :<%# Eval("Submit_date") %></br>
                <%# Eval("Revi_text") %></br>   
                <a href="" target="">回复评论</a></br></br>
            </ItemTemplate>
        </asp:DataList>
        <asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod=
"GetReviewAll" TypeName="BlogBLL.ReviewManager">
        </asp:ObjectDataSource>
    </div>
</asp:Content>
```

(7) 在发表日志页面,需要先判断 Session["user_id"] 是否为 null,从而判断该用户是否为登录用户。需要注意第三方控件 FCKeditor(在线文本编辑器)的使用,使用的详细说明可参照任务 4.3。

发表日志页面 IssueArticleFCKeditor.aspx 实现代码如下,运行效果如任务 4.3 中的图 4-7 所示。

```
<%@ Page Language="C#" AutoEventWireup="true" MasterPageFile="~/MicroBlog.Master"
    Codebehind="IssueArticleFCKeditor.aspx.cs" Inherits="BlogWebApp.
IssueArticleFCKeditor" %>

<%@ Register Assembly="FredCK.FCKeditorV2" Namespace="FredCK.FCKeditorV2" TagPrefix=
"FCKeditorV2" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" Text="日志标题"></asp:Label>
        <asp:TextBox ID="txt_title" runat="server"></asp:TextBox><br />
        <asp:Label ID="Label2" runat="server" Text="日志正文"></asp:Label><br />
        <FCKeditorV2:FCKeditor ID="FCKeditor1" runat="server" Height="280px">
        </FCKeditorV2:FCKeditor>
        <asp:Button ID="Button1" runat="server" Text="发表文章" OnClick="Button1_Click" />
```

```
    </div>
</asp:Content>
```

在 IssueArticleFCKeditor. aspx. cs 中编写代码如下：

```
...
using BlogModels;
using BlogBLL;
namespace BlogWebApp
{
    public partial class IssueArticleFCKeditor : System.Web.UI.Page
    {
        protected void Button1_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                LogArticle article = new LogArticle();
                //获取日志标题、日志内容
                article.Arti_title = txt_title.Text.Trim().ToString();
                article.Arti_text = FCKeditor1.Value.Trim().ToString();
                //阅读次数初始值为 0
                article.Arti_click = 0;
                //发表日志时间
                article.Submit_date = DateTime.Now;
                if (Session["user_id"] == null) //用户没有登录
                {
                    Response.Write("<script>alert('请先登录')</script>");
                }
                else
                {
                    //获取发表日志用户的 User_id
                    string UserId = Session["user_id"].ToString();
                    article.User_id = Convert.ToInt32(UserId);
                    if (ArticleManager.AddArticle(article, UserId) != 0)
                    {
                        Response. Redirect ( " ~/DefaultArticleAllRepeat. aspx ");
Response.Write("<script>alert('文章发表成功!')</script>");
                    }
                    else
                    {
                        Response.Write("<script>alert('Sorry,文章提交失败!')</script>");
                    }
                }
            }
        }
    }
}
```

(8) 在用户退出登录时，需清空 Session 对象。在退出登录页面 Logout. aspx. cs 中编写代码如下：

```
namespace BlogWebApp.Login
{
    public partial class Logout : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Session["user_id"] = null;     //清空 Session 对象,退出登录
            Response.Redirect("~/Login/Login.aspx");     //返回到登录页面
        }
    }
}
```

(9) 对于 Admin 文件夹,当前任务尚未设置任何访问限制,允许所有用户访问。

ArticleManagerGridView.aspx 是日志文章的后台管理页面,参考代码如下,运行效果如图 9-5 所示。

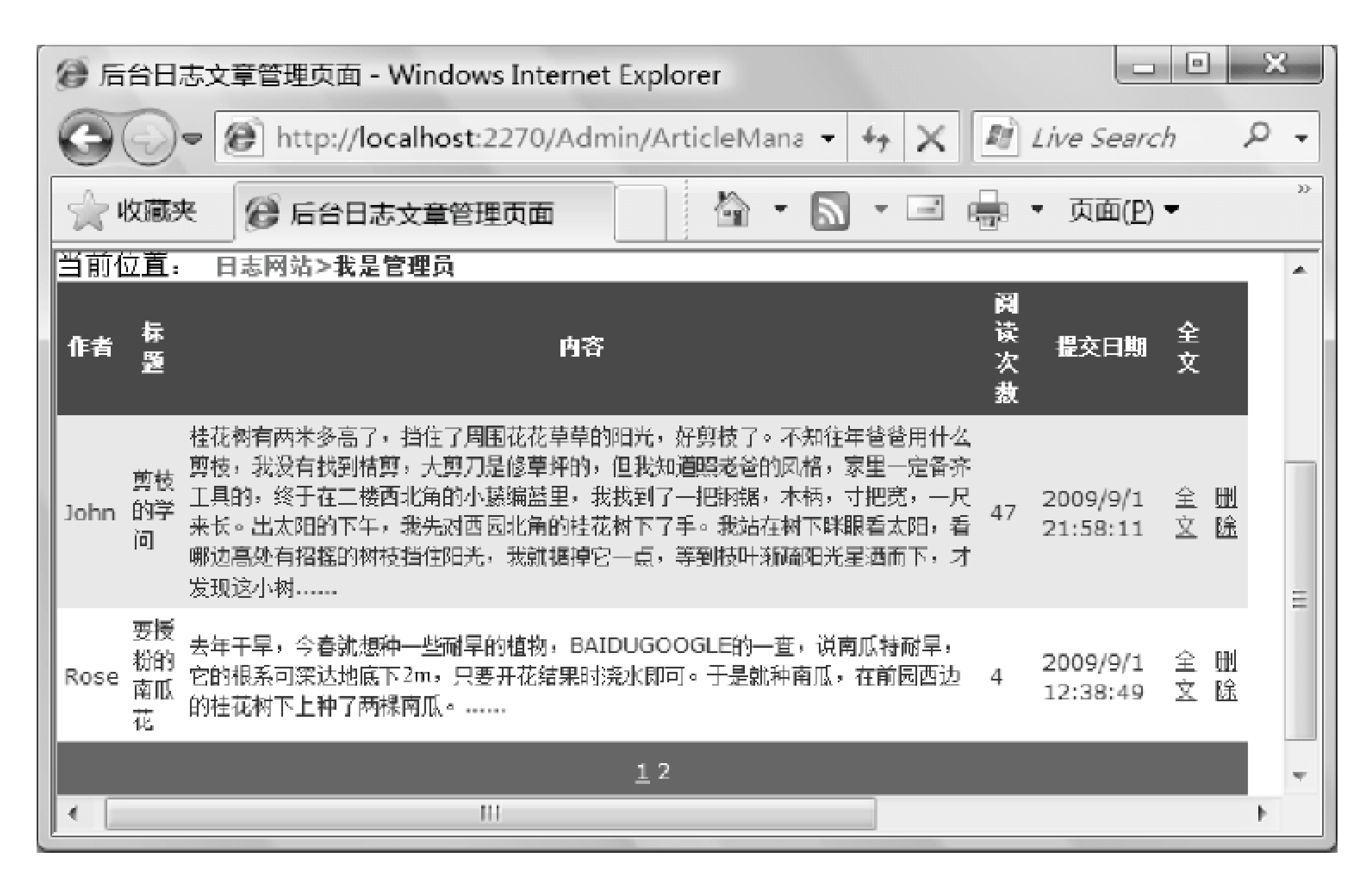


图 9-5　后台日志文章管理页面

```
<%@ Page Language = "C#" AutoEventWireup = "true" MasterPageFile = "~/MicroBlog.Master"
    Codebehind = " ArticleManagerGridView. aspx. cs " Inherits = " BlogWebApp.
ArticleManagerGridView" %>

<asp:Content ID = "Content1" ContentPlaceHolderID = "ContentPlaceHolder1" runat = "server">
    <div>
        <asp:GridView ID = "GridView1" runat = "server" AllowPaging = "True" AutoGenerateColumns =
"False" DataSourceID = "ObjectDataSource1" DataKeyNames = "Arti_id" CellPadding = "4" ForeColor =
"#333333" GridLines = "None" PageSize = "3">
            <Columns>
                <asp: BoundField DataField = " Arti _ id " HeaderText = " Arti _ id "
SortExpression = "Arti_id" Visible = "False" />
                <asp: BoundField DataField = " User _ id " HeaderText = " User _ id "
SortExpression = "User_id" Visible = "False" />
                <asp: BoundField DataField = " User _ name " HeaderText = " 作 者 "
```

```
SortExpression="User_name" />
                    <asp:BoundField DataField="Arti_title" HeaderText="标题"
SortExpression="Arti_title" />
                    <asp:TemplateField HeaderText="内容" SortExpression="Arti_text">
                        <ItemTemplate>
                            <%# GetPartString(Eval("Arti_text").ToString(),192) %>
                        </ItemTemplate>
                    </asp:TemplateField>
                    <asp:BoundField DataField="Arti_click" HeaderText="阅读次数"
SortExpression="Arti_click" />
                    <asp:BoundField DataField="Arti_review" HeaderText="评论数"
SortExpression="Arti_review" Visible="False" />
                    <asp:BoundField DataField="Submit_date" HeaderText="提交日期"
SortExpression="Submit_date" />
                    <asp:HyperLinkField HeaderText="全文" NavigateUrl="~/
ArticleDetailByArticleId.aspx" Text="全文" DataNavigateUrlFields="arti_id"
DataNavigateUrlFormatString="ArticleDetailByArticleId.aspx?articleid={0}" />
                    <asp:CommandField ShowDeleteButton="True" />
                </Columns>
                <RowStyle BackColor="#E3EAEB" />
                <FooterStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
                <PagerStyle BackColor="#666666" ForeColor="White" HorizontalAlign="Center" />
                <SelectedRowStyle BackColor="#C5BBAF" Font-Bold="True" ForeColor="#333333" />
                <HeaderStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
                <EditRowStyle BackColor="#7C6F57" />
                <AlternatingRowStyle BackColor="White" />
            </asp:GridView>
            <asp:ObjectDataSource ID="ObjectDataSource1" runat="server" DataObjectTypeName=
"BlogModels.LogArticle"
                DeleteMethod="DeleteArticle" SelectMethod="GetArticleAll" TypeName=
"BlogBLL.ArticleManager" UpdateMethod="UpdateArticle">
            </asp:ObjectDataSource>
        </div>
</asp:Content>
```

在 ArticleManagerGridView.aspx.cs 中编写代码如下：

```
...
using BlogModels;
using BlogBLL;
namespace BlogWebApp
{
    public partial class ArticleManagerGridView : System.Web.UI.Page
    {
        /// <summary>
        /// 截取文章的前面部分内容
        /// </summary>
        public static string GetPartString(string article_text, int num)
        {
            return ArticleManager.GetPartString(article_text, num);
```

```
        }
    }
}
```

9.1.9　小结

(1) 数据库连接属于应用程序级的配置,所以连接字符串通常是设置在站点的 Web.config 文件中的。在 Web.config 的<connectionStrings></connectionStrings>配置节中,也可以保存多个数据库连接的配置信息(使用多个 add 标记即可)。

(2) 表示层中的截取文章部分内容(截取字符串)等通用方法,通常情况下应该是放在单独创建的如 CommonHandler.cs 类中并放置在 App_Code 目录下,读者可以自行尝试。

(3) 限于篇幅,在上述所列代码中省略了发表日志页面、发表评论页面中输入控件的非空验证。数据库表中的某些 null 值将导致“未将对象引用设置到对象实例”等异常。切记务必给上述页面控件加上非空验证。

9.1.10　思考与练习

在每一个评论的后面添加一个“回复评论”超链接,当单击“回复评论”超链接时,显示一个可以添加回复的表单并将回复信息与该条评论一起发布显示。

提示: 可以增加一回复信息的数据库表。

任务 9.2　网站的安全认证与授权

任务目标

(1) 了解 Web.config 应用程序配置文件的文件结构以及它的网站安全性配置策略。

(2) 了解 ASP.NET 身份认证的基本方式以及它的用户授权机制。

(3) 掌握设置基于表单的身份验证和页面授权的方法。

9.2.1　网站安全性配置概述

在上一任务中,当用户请求发表日志时,是用 Session 来记录、判断用户是否为登录用户的:在用户登录页面,当用户登录成功时从数据库获取与其用户名、密码相对应的 user_id 存入 Session 中,然后在需要的每一个页面判断 Session["user_id"]是否为 null,否则断定该用户没有登录;在该用户退出时,令其 Session["user_id"] = null。这样很直观,但无疑使代码冗长并增加了编码的复杂性。

其实在 ASP.NET 的安全体系架构中,有着更简捷的实现方案:那就是简单的配置 Web.config 文件,实现应用程序和页面文件的认证与授权。

所谓认证,就是身份验证,要求用户输入用户名和密码进行登录验证后方可进入应用程序或页面文件;而所谓授权,就是对进入应用程序的用户授予访问哪些页面或资源的权利。这有点像进学校的图书馆:进入校图书馆的必须是本校的教工或学生,这是已经

经过认证的；而进入图书馆大楼以后，一般用户能自由出入阅览室、借阅室；若要进藏书库、入编目室则必须要得到许可或授权。

一个站点就是一个应用程序。Web.config 是应用程序配置文件。一个应用程序的各级目录下都可以有 Web.config 文件，每一个 Web.config 只对当前目录及其子目录起作用。每一个子目录都可以继承上一级目录的配置并覆盖其中相同的选项。

Web.config 是一个 XML 文件。可在"解决方案资源管理器"面板中右击 MicroBlog 项目的 Web 层 BlogWebApp，在弹出的快捷菜单中选择"添加"→"新建项"命令，在打开的"添加新项-BlogWebApp"对话框中，选择 "Web 配置文件"选项，Web.config 默认的文件格式如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<!--
    注意：除了手动编辑此文件以外，还可以使用
    Web 管理工具来配置应用程序的设置.可以选择 Visual Studio 中的
    "网站" - >"Asp.Net 配置"命令.
    设置和注释的完整列表在
    machine.config.comments 中，该文件通常位于
    \Windows\Microsoft.Net\Framework\v2.x\Config 中
 -->
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
        <!--
            设置 compilation debug = "true" 将调试符号插入
            已编译的页面中.但由于这会
            影响性能，因此只在开发过程中将此值
            设置为 true.
         -->
        <compilation debug = "false" />
        <!--
            通过 <authentication> 节可以配置 ASP.NET 使用的
            安全身份验证模式，
            以标识传入的用户.
         -->
        <authentication mode = "Windows" />
        <!--
            如果在执行请求的过程中出现未处理的错误，
            则通过 <customErrors> 节可以配置相应的处理步骤.具体说来，
            开发人员通过该节可以配置
            要显示的 html 错误页
            以代替错误堆栈跟踪.

        <customErrors mode = "RemoteOnly" defaultRedirect = "GenericErrorPage.htm">
            <error statusCode = "403" redirect = "NoAccess.htm" />
            <error statusCode = "404" redirect = "FileNotFound.htm" />
        </customErrors>
```

```
        -->
    </system.web>
</configuration>
```

可以看到，Web.config 文件的根节点是<configuration>，所有的配置信息都位于<configuration></configuration>根节点之中。其中：

<appSettings></appSettings> 是应用程序设置节，用来定义应用全局常量如数据库连接字符串常量等信息；

<authentication></authentication>可称为认证节，用来设置应用程序的身份验证模式。没有经过身份验证访问应用程序的用户称为匿名用户，在默认情形下，应用程序是允许匿名访问的；

<authorization></authorization>可称为授权节，用于设置允许(allow)或拒绝(deny)用户或用户组访问特定页面资源的权利；

<customErrors></customErrors>是自定义错误节，用来指定发生错误时可以重定向的页面的 URL。这样可以自行编写出错时的信息显示页面，以免程序发生错误时系统抛出的异常直接被用户看见。

9.2.2　ASP.NET 身份验证模式

ASP.NET 支持以下 3 种模式的身份验证。

(1) 基于 Microsoft Windows 的身份验证。Windows 认证要求每一个用户在应用程序所在的服务器上拥有一个用户账号，并在 Web.config 文件中设置如下代码：

```
<system.web>
    <authentication mode = "Windows" />
</system.web>
```

(2) 基于 Microsoft Passport 的身份验证。Passport 认证是 Microsoft 收费的一种注册通行证服务。目前国内网站较少使用。

(3) 基于 Forms 的身份验证。Forms 认证需要在应用程序中增加一个登录页面。当用户访问应用程序的受限页面时，将被自动引导到登录页面。用户输入的用户名和密码经数据库验证后倘若正确，表示通过认证，在客户端将自动创建一个以用户名为参数的认证 Cookie，并重定向到用户刚才请求的页面。

例如，倘若希望所有访问管理员后台(Admin 文件夹下的所有页面)的用户都必须先到登录页面 Login.aspx 去登录以验明身份，则可以在 Web.config 文件中设置如下代码：

```
<authentication mode = "Forms">
    <forms name = "userCookie" loginUrl = "~/Login/Login.aspx" timeout = "60"/>
</authentication>

<location path = "Admin">
    <system.web>
        <authorization>
            <allow roles = "admin"/>
```

```
            <allow users = "Rose"/>
            <deny users = " * "/>
        </authorization>
    </system.web>
</location>
```

说明：

① 在以上授权节中，授权角色(即用户组)admin 和用户 Rose 可以访问 Admin 文件夹下的文件。<deny users=" * "/>表示拒绝所有用户，<allow users="?"/>表示允许匿名用户。注意通配符的用法。

② 在以上认证节中，mode 设置了基于 Forms 的身份验证。name 属性用来指定认证 Cookie，Cookie 名称任意；loginUrl 属性指定将被自动引导的登录页面 URL；timeout 属性指定认证 Cookie 的到期时间(分钟)。

基于 Forms 的认证使用 Cookie 来保持页面之间的状态。因为设置简单，是迄今为止广泛使用的认证模式。

9.2.3 Blog 网站的安全性配置策略

(1) 继续创建日志网站。站点目录结构、数据库文件及主要页面文件可以参考任务 9.1。站点首页是 DefaultArticleAllRepeat.aspx。

(2) 在 Web.config 中设置页面的授权。

对于匿名用户、认证用户、管理员账号用户等，分别授予允许访问或拒绝访问特定页面资源的权利：

对于包括匿名用户在内的所有用户，授予允许访问首页(浏览日志)的权利；授予允许访问 Login 登录文件夹(用户登录、新用户注册)的权利；

对于认证用户，授予允许访问发表日志页面的权利；

对于管理员用户，则授予其允许访问 Admin 管理员文件夹的权利。

注意：对于 Admin 管理员文件夹，同时还要设置拒绝匿名访问，因为在默认情形下，应用程序是允许匿名访问的。

在 MicroBlog 项目 Web 站点下，Web.config 应用程序配置文件的代码如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
<configuration>
  <appSettings/>
  <connectionStrings>
      <add name = " userlogConnectionString " connectionString = "Data Source = .\SQLEXPRESS;
AttachDbFilename = |DataDirectory|\userlog.mdf;Integrated Security = True;User Instance = True"
providerName = "System.Data.SqlClient" />
  </connectionStrings>

  <system.web>
     <!--
            设置 compilation debug = "true" 将调试符号插入已编译的页面中.但由于这会
            影响性能,因此只在开发过程中将此值设置为 true.
```

```
      -->
      <compilation debug="true"/>
      <!--
            通过 <authentication> 节可以配置 ASP.NET 使用的安全身份验证模式,
            以标识传入的用户.
      -->
    <authentication mode="Forms">
      <forms name="AuthCookie" loginUrl="~/Login/Login.aspx"/>
    </authentication>

     <authorization>
      <allow users="?"/>
    </authorization>
  </system.web>

  <location path="Login">
    <system.web>
      <authorization>
        <allow users="*"/>
      </authorization>
    </system.web>
  </location>

  <location path="Admin">
    <system.web>
      <authorization>
        <allow users="Rose,John"/>
        <deny users="*"/>
      </authorization>
    </system.web>
  </location>

  <system.web>
    <!--
     如果在执行请求的过程中出现未处理的错误,则通过 <customErrors> 节
          可以配置相应的处理步骤.具体说来,开发人员通过该节可以配置要显示的
          html 错误页以代替错误堆栈跟踪.
     -->
     <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
      <error statusCode="403" redirect="NoAccess.htm" />
      <error statusCode="404" redirect="FileNotFound.htm" />
      <error statusCode="500" redirect="ServerError.htm" />
     </customErrors>
    </system.web>
  </configuration>
```

可以看到,这里用<allow users="Rose,John"/>,一个逗号分隔的用户名列表,来代替了 roles 角色(用户组)的授权设置:授权用户 Rose、John 可以访问 Admin 文件夹。Forms 验证没有为角色验证提供直接支持,所以这是最简单的给用户组授权的方法。

(3) 在登录页面的 Login. aspx. cs 中新增的几行代码,用以在通过登录的数据库验证以后给用户发放凭证 Cookie。修改后的 Login. aspx. cs 代码如下:

```
...
using BlogModels;
using BlogBLL;
namespace BlogWebApp.Login
{
    public partial class Login : System.Web.UI.Page
    {
        protected void Button1_Click(object sender, EventArgs e)
        {
            if (Page.IsValid)
            {
                User user = new User();
                user.User_name = txt_name.Text.Trim();
                user.User_pw = txt_pw.Text.Trim();

                //判断登录用户名和密码是否正确
                if (UserManager.UserLogin(user, out user))
                {
                    //保存发表日志时需要的用户主键标识
                    Session["user_id"] = user.User_id;
                    //通过登录的数据库验证,并在客户端创建一个以登录名为参数的认证 Cookie
                    FormsAuthentication.SetAuthCookie(txt_name.Text, false);
                    //重定向到管理员页面
                    Response.Redirect("~/Admin/ArticleManagerGridView.aspx");
                }
                else
                {
                    txt_msg.Text = "请查证并输入有效的用户名和密码!";
                    txt_name.Text = string.Empty;
                    txt_pw.Text = string.Empty;
                }
            }
        }
    }
}
```

当用户请求管理员页面(比如日志文章管理页面 ArticleManagerGridView. aspx)时,将被自动引导到 Login. aspx 页面登录;在 Login. aspx. cs 文件中根据用户输入的用户名和密码在数据库 userlog. mdf(userinfo 表)中寻找匹配记录。倘若通过数据库验证,系统将自动在客户端创建一个以用户名为参数的认证 Cookie 并可重定向到刚才用户请求的(管理员)页面。false 表示关闭浏览器后不保留认证 Cookie。

上述新增的两行代码也可以合并成一行,并且是重定向到用户刚才请求的页面,代码如下:

```
FormsAuthentication.RedirectFromLoginPage(txt_name.Text, false);
```

(4) 当用户退出登录时,可以使用 FormsAuthentication. SetAuthCookie(null, false);或者 FormsAuthentication. SignOut();删除认证 Cookie。

修改 Logout. aspx. cs 中 Page_Load 事件代码如下:

```
namespace BlogWebApp.Login
{
    public partial class Logout : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            Session["user_id"] = null;                    //清空 Session 对象
            FormsAuthentication.SignOut();                //删除认证 Cookie,退出登录
            Response.Redirect("~/Login/Login.aspx");      //返回到登录页面
        }
    }
}
```

9.2.4 小结

在 Web. config 中还可以设置网站安全性等其他众多配置选项。例如,可以使用 ASP.NET 提供的网站管理工具(而不需要手动编辑)来创建或编辑配置文件:单击“解决方案资源管理器”面板中的“ASP.NET 配置” 按钮,即可进入站点管理工具的欢迎页面。它提供了一个 Web 界面,用于管理当前站点的配置设置。

网站管理工具提供了“主页”、“安全”、“应用程序”、“提供程序”这 4 个选项卡。

使用“安全”选项卡可以管理访问规则、可以创建和管理用户与角色(用户组)。

使用“应用程序”选项卡可以管理与网站有关的各种设置,其中包括:应用程序设置、SMTP 设置、调试和跟踪设置等。

使用“提供程序”选项卡可以指定存储网站成员和角色信息的数据库提供程序。在默认情况下,网站管理工具在网站的 App_Data 文件夹中创建一个本地 SQL Server Express 数据库用以存储用户信息;如果要将用户信息存储在其他数据库中,可以使用“提供程序”选项卡选择其他提供程序。

首次使用网站管理工具时,如果 Web. config 文件不存在,网站管理工具将创建该文件;当更改默认配置时,将修改 Web. config 文件。对于大多数设置而言,在网站管理工具中进行的更改都将立即生效并反映在 Web. config 文件中。

读者可以参见 ASP.NET 网站管理工具的帮助文件得到更详细的说明。

9.2.5 思考与练习

为 0931 新闻网站配置基于 Forms 的身份验证与页面授权。

第 10 章　ASP.NET AJAX

任务 10.1　使用 ASP.NET AJAX Extensions 优化新闻搜索页

任务目标

(1) 熟悉 ASP.NET AJAX Extensions 控件。
(2) 会用 ASP.NET AJAX Extensions 实现页面局部更新。

10.1.1　ASP.NET AJAX 简介

前面介绍的以服务器端为中心的 ASP.NET Web 应用程序有一个很大的缺陷，就是它需要频繁地刷新页面，页面响应速度慢。传统的 Web 应用允许用户填写表单(Form)，当提交表单时就向 Web 服务器发送一个请求。服务器接收并处理传来的表单，然后返回一个新的网页。这个做法浪费了许多带宽，因为在前后两个页面中的大部分 HTML 代码往往是相同的。由于每次应用的交互都是向服务器发送请求并获得新的网页，应用的响应时间就依赖于服务器的响应时间。这导致了用户界面的响应比本地应用慢得多。

与此不同，使用 AJAX 能在不刷新整个页面的前提下维护数据。这使得 Web 应用程序更为迅捷地响应用户交互，并避免了在网络上发送那些没有改变的信息。AJAX (Asynchronous JavaScript and XML，异步 JavaScript 和 XML)是一种创建交互式网页应用的网页开发技术。事实上，AJAX 不是指一种单一的技术，而是有机地利用了一系列相关的技术。它使用 XHTML+CSS 来表示信息；使用 JavaScript 操作 Document Object Model 进行动态显示及交互；使用 XML 和 XSLT 进行数据交换及相关操作；使用 XMLHttpRequest 对象与 Web 服务器进行异步数据交换；使用 JavaScript 将所有的东西绑定在一起。AJAX 应用可以仅向服务器发送并取回必需的数据，它使用 SOAP 或其他一些基于 XML 的 Web Service 接口，并在客户端采用 JavaScript 处理来自服务器的响应。因为在服务器和浏览器之间交换的数据大量减少，所以就能看到响应更快的应用。同时由于很多的处理工作可以在发出请求的客户端机器上完成，因此 Web 服务器的处理时间也减少了。简单地说，AJAX 就是一种实现无刷新效果，增强用户体验的网页技术。

微软推出的 ASP.NET 的 AJAX 框架——ASP.NET AJAX，将 AJAX 技术集成到了 ASP.NET 平台中，成为 ASP.NET 的一种扩展技术。ASP.NET AJAX 是一个免费框架，是目前对 AJAX 技术最完备的封装，其丰富的组件可以让用户非常方便地利用面向

对象的思想实现 AJAX 网页技术，快速地创建有效率及交互式的、能在各种流行的浏览器工作的 Web 应用程序。ASP.NET AJAX 框架由两部分组成：客户端脚本库和服务器组件。客户端脚本库(ASP.NET AJAX 框架的客户端部分)将跨浏览器的 ECMAScript (JavaScript)技术和动态的 HTML(DHTML)技术结合在一起，并与基于 ASP.NET 服务器的开发平台集成。服务器组件(ASP.NET AJAX 框架的服务器端部分)包括服务器端控件 ASP.NET 2.0 AJAX Extensions 和以 ASP.NET AJAX Control Toolkit 为代表的第三方控件，是对 ASP.NET AJAX 框架的客户端脚本库与应用在服务器端的封装。它对 JavaScript 进行了非常巧妙的面向对象方面的扩展，以提供对客户端面向对象编程的支持。这使得用户在开发 ASP.NET AJAX Web 应用程序时，只需要关注 ASP.NET AJAX 服务器端控件，就像关注其他 ASP.NET 控件一样，根本不需要关注 JavaScript，甚至不需要了解 JavaScript 也可以实现优秀的 AJAX 效果。

ASP.NET AJAX Web 应用程序与完全基于服务器的 Web 应用程序相比，具有很多优点。它可以提供如下功能。

(1) 增强的效率，因为网页的大部分处理工作是在浏览器中执行的。

(2) 熟悉的 UI 元素，如进度指示器、工具提示和弹出窗口。

(3) 部分页更新，只刷新已发生更改的网页部分。

(4) 客户端与用于 Forms 身份验证的 ASP.NET 应用程序服务、角色和用户配置文件的集成。

(5) 自动生成的代理类，可简化从客户端脚本调用 Web 服务方法的过程。

(6) 一个框架，可自定义服务器控件以包含客户端功能。

(7) 对大部分流行的和常用的浏览器的支持，其中包括 Microsoft Internet Explorer、Mozilla Firefox 和 Apple Safari。

10.1.2 安装 ASP.NET AJAX Extensions

ASP.NET AJAX Extensions 是开发 ASP.NET AJAX 程序最重要的安装包，是 ASP.NET AJAX 的核心，包括最重要和最基本的一些控件和功能。对于这个安装包中的内容，微软公司为其提供了完善的技术支持，在 MSDN 中也有对应的开发文档等。

本教程使用的 ASP.NET 2.0 AJAX Extensions 1.0 可从微软 ASP.NET 官方网站免费下载。具体步骤为：进入 http://www.asp.net/ajax/downloads/archive/，单击链接 Download ASP.NET AJAX Extensions v1.0，下载文件名为 ASPAJAXExtSetup.msi 的安装程序并运行。安装完毕后，在 Visual Studio 2005 中选择“新建项目”菜单会出现 ASP.NET AJAX-Enablled Web Application 项目模板，用它即可新建经过基本配置的 ASP.NET AJAX Web 应用程序。

10.1.3 ASP.NET AJAX Extensions 控件简介

ASP.NET 2.0 AJAX Extensions 控件包括以下 5 种。

1. 脚本管理控件 ScriptManager

ASP.NET AJAX 的核心就是 ScriptManager 控件，该控件用于管理支持 ASP.NET

AJAX 页的客户端脚本。任何一个使用 ASP.NET AJAX 的页面都需要加入 ScriptManager，但每个页面中有且只能有一个 ScriptManager，且必须出现在任何需要它的控件之前。它提供了实现 ASP.NET AJAX 功能需要的基本的脚本。

2. 局部更新控件 UpdatePanel

UpdatePanel 控件用于实现页面局部更新(即在页面回传过程中刷新网页的选定部分，而不是刷新整个网页)。UpdatePanel 控件相当于一个容器，可以把页面中需要更新的部分内容放置其中。例如，把一个 GridView 控件拖到 UpdatePanel 中，则 GridView 的分页、排序操作都会通过 AJAX 请求实现页面局部更新，而不是每个操作都引起整个页面更新。这种更新方式称为“部分页更新”。部分页更新相比整页刷新可节省系统开销，也会让用户感觉更流畅。

这里需要指出的是，用 UpdatePanel 控件进行部分页更新时，提交给服务器的数据跟一般的回传没有什么区别，包括 ViewState 中的数据在内的所有数据被传回服务器。不同的地方在于从服务器只返回部分更新部分的数据。UpdatePanel 控件的回传方式被分为两种，异步回传和常规回传，异步回传引发 UpdatePanel 的更新，常规回传引发整个页面的更新。

下面介绍 UpdatePanel 的几个常用属性。

(1) ContentTemplate

ContentTemplate 属性用来定义 UpdatePanel 的内容，在其内部可以放 ASP.NET 控件或 HTML 元素。例如，在 ContentTemplate 内放置一个 Button 控件的代码为：

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
         <asp:Button ID="Button1" runat="server" Text="Button" />
    </ContentTemplate>
</asp:UpdatePanel>
```

(2) Triggers

UpdatePanel 控件有两种 Triggers 属性，即 AsyncPostBackTrigger 和 PostBackTrigger。其中，AsyncPostBackTrigger 属性用来指定某个服务器控件，并将该控件触发的服务器事件作为该 UpdatePanel 控件的异步更新触发器；而 PostBackTrigger 属性用来指定在 UpdatePanel 控件中的某个服务器控件，它引发常规回传、整页更新。下面的代码举例说明了 Triggers 属性的使用方式。

```
<asp:Button ID="Button1" runat="server" Text="异步回传" OnClick="Button1_Click"/>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <asp:Button ID="Button2" runat="server" Text="常规回传" OnClick="Button2_Click"/>
    </ContentTemplate>
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
        <asp:PostBackTrigger ControlID="Button2" />
    </Triggers>
</asp:UpdatePanel>
```

其中 ControlID 和 EventName 属性分别用来指定控件 ID 和控件事件，若没有明确指定 EventName 的值，则自动采用控件的默认值，比如 Button 就是 Click。在上面的代码中，Button1 采用异步回传的方式，而 Button2 采用常规回传的方式。Button1 位于 UpdatePanel 之外，在默认情况下单击该按钮会产生常规回传，现在在 UpdatePanel 中添加了 AsyncPostBackTrigger 属性指定 Button1，单击该按钮产生的却是异步回传。Button2 位于 UpdatePanel 中，在默认情况下单击该按钮会产生异步回传，现在在 UpdatePanel 中添加了 PostBackTrigger 属性指定 Button2，单击该按钮产生的却是常规回传。

（3）UpdateMode

UpdateMode 属性表示 UpdatePanel 的更新模式，可取值为 Always（总是更新）或 Conditional（有条件更新），默认值为 Always。Always 表示不管有没有 Trigger，其他控件都将更新该 UpdatePanel；Conditional 表示只有当前 UpdatePanel 的 Trigger 被指定或 ChildrenAsTriggers 属性为 true 时，当前 UpdatePanel 中控件引发的异步回传或者常规回传，或是服务器端调用 Update()方法才会引发更新该 UpdatePanel。

有时需要为一个页面的多个需要局部更新的区域加上不同的 UpdatePanel。由于 UpdatePanel 默认的 UpdateMode 属性是 Always，如果页面上有一个局部更新被触发，则所有的 UpdatePanel 都将更新，这是不愿看到的，只需要 UpdatePanel 在它自己的触发器触发的时候更新就可以了，所以需要把 UpdateMode 设置为 Conditional。这个值的设定可以使各个 UpdatePanel 在各种合适的时机更新。

例如，某页面包括两个 UpdatePanel。UpdatePanel1 用来输入数据，其中包括几个输入框、一个“添加”按钮和一个“取消”按钮。UpdatePanel2 用来显示数据，里面放置一个 GridView。每一个 UpdatePanel 的 UpdateMode 属性都设置为 Conditional，并将 UpdatePanel2 的 Triggers 指定由“添加”按钮引发异步回传。这时，当单击“取消”按钮时，只有 UpdatePanel1 刷新，当单击“添加”按钮时，两个 UpdatePanel 都刷新。

（4）ChildrenAsTriggers

ChildrenAsTriggers 属性用来指示 UpdatePanel 中直接子控件的异步回传是否会引发 UpdatePanel 的更新。ChildrenAsTriggers 属性可取值为 True 或 False，设置为 True 时更新，否则不更新，默认为 True。如果此属性设置为 False，UpdatePanel 控件的 UpdateMode 属性就必须设置为 Conditional，否则系统会抛出异常。

表 10-1 所示将 UpdatePanel 的 UpdateMode 和 ChildrenAsTriggers 属性取值组合情况作出总结。

注意：*在假设某 UpdatePanel 的 ID 为 up1，并不设置 AsyncPostBackTrigger 和 PostBackTrigger 两个 Triggers 属性的前提下。*

UpdatePanel 更新情况总结如下：如果 UpdatePanel 控件不在另一个 UpdatePanel 控件的内部，则面板的更新将取决于 UpdateMode 和 ChildrenAsTriggers 属性以及触发器集合的设置；如果某个 UpdatePanel 控件位于另一个 UpdatePanel 控件内部，则子面板会自动随父面板一同更新。

出现下列情况时，会更新 UpdatePanel 控件的内容。

表 10-1 UpdateMode 和 ChildrenAsTriggers 属性取值组合情况总结

属性取值组合	说明
UpdateMode="Always" ChildrenAsTriggers="True"	up1 内部控件可对 up1 内容实现异步回传,局部更新 其他 UpdatePanel 内部控件可对 up1 内容实现异步回传,局部更新 UpdatePanel 之外的控件不可对 up1 内容实现异步回传。UpdatePanel 之外的控件引发常规回传,整页更新。除非用 AsyncPostBackTrigger 指定 UpdatePanel 之外的控件对 up1 内容实现异步回传,局部更新
UpdateMode="Always" ChildrenAsTriggers="False"	不允许这种设置
UpdateMode="Conditional" ChildrenAsTriggers="True"	up1 内部控件可对 up1 内容实现异步回传,局部更新 其他 UpdatePanel 内部控件不可对 up1 内容实现异步回传。除非用 AsyncPostBackTrigger 指定其他 UpdatePanel 内部控件对 up1 内容实现异步回传,局部更新 UpdatePanel 之外的控件不可对 up1 内容实现异步回传。UpdatePanel 之外的控件引发常规回传,整页更新。除非用 AsyncPostBackTrigger 指定 UpdatePanel 之外的控件对 up1 内容实现异步回传,局部更新
UpdateMode="Conditional" ChildrenAsTriggers="False"	up1 内部控件不可对 up1 内容实现异步回传。除非用 AsyncPostBackTrigger 指定 up1 内部控件对 up1 内容实现异步回传,局部更新 其他 UpdatePanel 内部控件不可对 up1 内容实现异步回传。除非用 AsyncPostBackTrigger 指定其他 UpdatePanel 内部控件对 up1 内容实现异步回传,局部更新 UpdatePanel 之外的控件不可对 up1 内容实现异步回传。UpdatePanel 之外的控件引发常规回传,整页更新。除非用 AsyncPostBackTrigger 指定 UpdatePanel 之外的控件对 up1 内容实现异步回传,局部更新

(1) 如果将 UpdateMode 属性设置为 Always,则每次从页上的任何位置执行回传时都会更新 UpdatePanel 控件的内容。回传包括来自其他 UpdatePanel 控件所包含的控件的异步回传,也包括来自 UpdatePanel 控件未包含的控件的回传。

(2) UpdatePanel 控件嵌套在另一个 UpdatePanel 控件中并且更新父面板时。

(3) 将 UpdateMode 属性设置为 Conditional 并且满足以下条件之一时。

① 显式调用 UpdatePanel 控件的 Update 方法。

② 通过使用 UpdatePanel 控件的 Triggers 属性定义为触发器的控件导致回传。在这种情况下,该控件显式触发面板内容的更新。该控件可以位于定义触发器的 UpdatePanel 控件的内部或外部。

③ 将 ChildrenAsTriggers 属性设置为 True 并且 UpdatePanel 控件的子控件导致回传。嵌套的 UpdatePanel 控件的子控件不会导致更新外部 UpdatePanel 控件,除非将子控件显式定义为触发器。

3. 更新进度条控件 UpdateProgress

UpdateProgress 控件用于为长时间的 AJAX 调用给用户提供反馈。在理想的情况下，AJAX 可以很好地工作。但是如果遇到服务器响应速度慢、网络速度不理想、复杂的数据库请求等原因，需要用户等待一段时间，此时 AJAX 则会让用户觉得没有任何的反馈，这是用户不希望看到的。通常的解决方案是，使用一个可视化的组件来告诉用户系统正在进行后台操作并且正在读取数据和内容。UpdateProgress 控件提供了这样的功能，它能够以一种友好的方式向用户显示页面和服务器正在进行交互，给用户提供反馈，告知用户操作正在进行，用户也就不必为此而感到无所适从了。UpdateProgress 控件运行时的表现形式完全可以通过其模板进行自定义，可以让 UpdateProgress 控件显示一个有意义的消息，或者提供一个取消按钮使用户可以取消操作。

4. 计时器控件 Timer 控件

Timer 控件用于在一个规定的时间间隔内引发时钟事件，对页面进行同步或异步周期性地更新。如果将 Timer 控件和 UpdatePanel 控件结合在一起使用，可以按照定义的时间间隔启用部分页更新，也可以使用 Timer 控件来发送整页。

Timer 控件的一般形式如下：

```
<asp:Timer ID = "Timer1" runat = "server" OnTick = "Timer1_Tick" Interval = "60000">
</asp:Timer>
```

它表示每隔 1 分钟引发服务端的时钟事件从而发起 AJAX 调用。Timer 控件的 Interval 属性用于指定时间间隔，单位是毫秒，默认值为 60000(60s)。Timer 控件的 Tick 事件用于指定所要触发的事件。

5. 脚本管理代理控件 ScriptManagerProxy

ScriptManagerProxy 控件允许内容页和用户控件等嵌套组件在父元素中已定义了 ScriptManager 控件的情况下将脚本和服务引用添加到网页。ScriptManagerProxy 和 ScriptManager 的功能很相似。但 ScriptManagerProxy 只是试图充当代理去调用 ScriptManager 的功能，从而保证了同一个页面实际上只存在一个 ScriptManager。

10.1.4 使用 ASP.NET AJAX Extensions 实现新闻搜索列表的局部刷新显示

借助 ASP.NET AJAX 控件，使用很少的客户端脚本或不使用客户端脚本就能创建丰富的客户端行为，如在异步回传过程中进行部分页更新和显示更新进度。不过，所有 ASP.NET AJAX 控件都需要 Web.config 文件中的特定设置才能正常运行。如果在 Visual Studio 2005 中，选择 ASP.NET AJAX-Enablled Web Application 项目模板新建一个 ASP.NET AJAX Web 应用程序，ASP.NET AJAX 相关的一些配置会自动添加到该 Web 应用程序的 Web.config 中。如果想在已有的 Web 应用程序中添加 ASP.NET AJAX 的功能，就需要手动修改 Web.config 文件。

下面对任务 6.3 中已经完成的新闻搜索页 NewsSearch.aspx 再进行优化，增加 ASP.NET AJAX Extensions 相关控件，实现新闻搜索列表的局部刷新显示。对于这个已存在的新闻网站，原 Web.config 中一些经过设置的值是要保留的。在配置 Web.config 时可

以进行如下操作。

(1) 将新闻网站原 Web. config 备份。

(2) 新建一个 ASP.NET AJAX Web 应用程序,将自动生成的 Web. config 中的代码复制并覆盖新闻网站中原 Web. config 的内容。

(3) 打开步骤(1)中的备份文件,将原 Web. config 中手动修改过的配置代码添加到新闻网站新的 Web. config 中去。

(4) 打开 News 解决方案,在“解决方案资源管理器”面板中,向 Web 项目添加对 System. Web. Extensions. dll 程序集的引用,如图 10-1 所示。

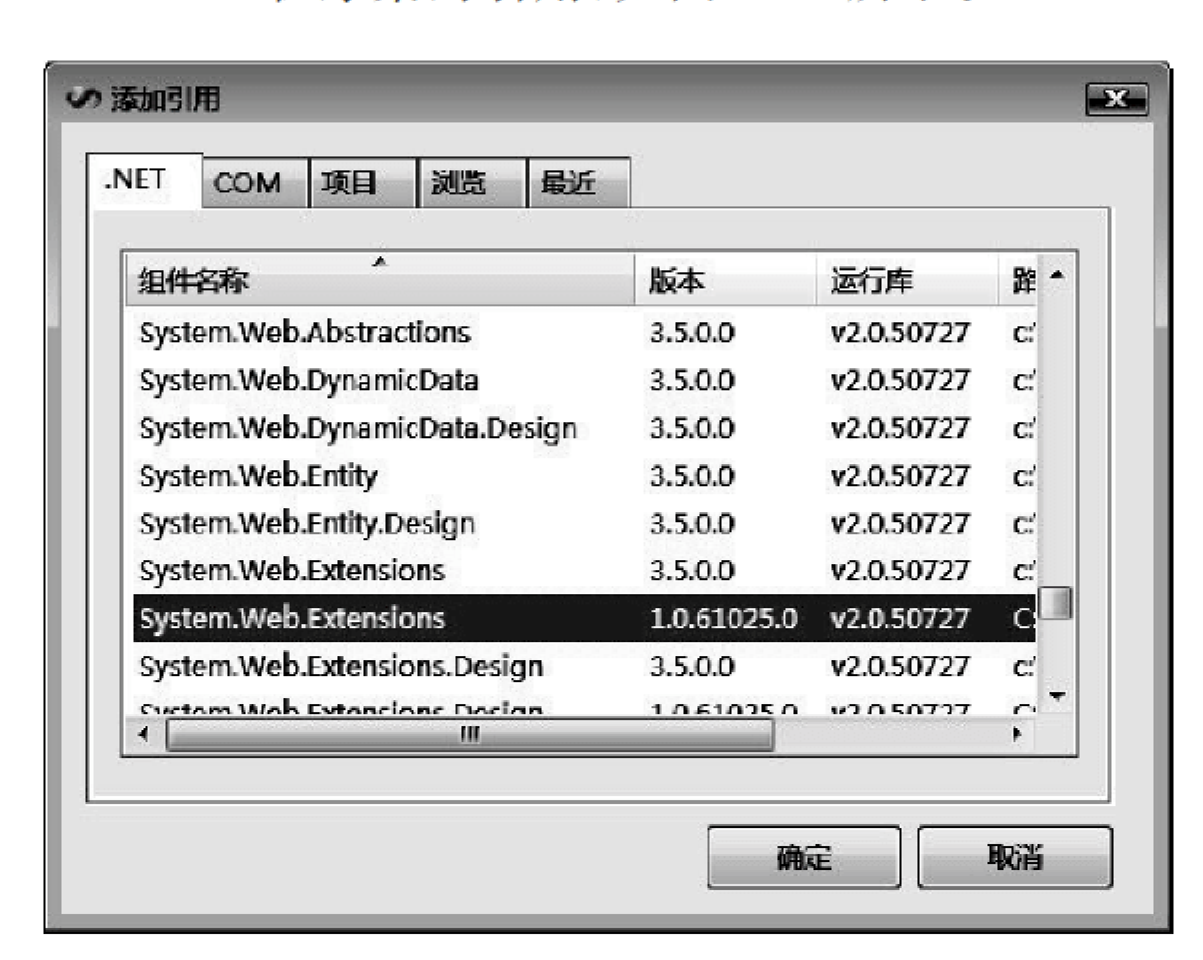


图 10-1 添加 System. Web. Extensions 引用

新闻搜索页的修改主要通过以下两步来完成。

1. 设置页面局部更新

设置页面局部更新的基本步骤如下:

(1) 添加 ScriptManager 控件。在将 UpdatePanel 控件添加到页面上之前,必须添加一个 ScriptManager 控件。UpdatePanel 控件依赖于 ScriptManager 控件来管理部分页更新。由于一个页面中只能放入一个 ScriptManager 控件,因此,若用母版页设计网页,可将 ScriptManager 控件放在母版页中,内容页就不需要再加入 ScriptManager 控件了。打开母版页 NewsPage. Master,切换到“设计”视图,将工具箱的 AJAX Extensions 选项卡中的 ScriptManager 控件拖放到页面 Form 开头部分。如果没有在母版页上加入 ScriptManager 控件,则必须在每一个会使用到 UpdatePanel 或其他 ASP.NET AJAX 控件的内容页面上加入 ScriptManager 控件。

(2) 添加 UpdatePanel 控件。打开内容页 NewsSearch. aspx,向页面添加 UpdatePanel 控件,并将排序、分页按钮及 Repeater 等需更新的列表内容全部移至 UpdatePanel 中。保存修改,然后按 Ctrl+F5 组合键在浏览器中查看页面。

通过比较可见,在原来的新闻搜索页中,单击排序或分页按钮发生回传时,页面会出现闪烁。而现在经过优化的页面在回传时,仅刷新 UpdatePanel 控件部分,不会发生页面

闪烁。

注意：对于不需要更新的内容，例如页面中固定不变的内容，就不要将其放在 UpdatePanel 控件中。目的是为了缩小需要更新的范围，从而有效地缩短更新的时间，让用户感觉页面反应很快。

2. 设置更新进度显示

若遇到网络延迟，这时候如果在页面上出现一个 GIF 动画，告诉用户稍等，等服务器处理完数据的时候 GIF 动画消失，就会让用户感觉体贴很多。可以使用 UpdateProgress 控件来实现这个功能。在请求 UpdatePanel 的新内容时，用 UpdateProgress 显示状态消息。

设置更新进度显示的基本步骤如下：

(1) 添加 UpdateProgress 控件。打开内容页 NewsSearch.aspx，切换到“设计”视图，将工具箱的 AJAX Extensions 选项卡中的 UpdateProgress 控件拖放到页面 UpdatePanel 控件的下方。

(2) 建立控件关联。选择 UpdateProgress 控件，并在“属性”窗口中将 AssociatedUpdatePanelID 属性设置为 UpdatePanel1。这会将 UpdateProgress 控件和先前添加的 UpdatePanel 控件相关联。

(3) 设置进度提示内容。在 UpdateProgress 控件的可编辑区域<ProgressTemplate></ProgressTemplate>模板中，插入文字“正在更新...”和一个 GIF 动画。保存修改，然后按 Ctrl+F5 组合键在浏览器中查看页面。如果在页面运行 SQL 查询和返回数据时出现延迟，则 UpdateProgress 控件将显示输入到 UpdateProgress 控件中的消息。

(4) 如果应用程序快速更新每页数据，则可能看不到页上的 UpdateProgress 控件的内容。UpdateProgress 控件支持 DisplayAfter 属性，该属性可以使得在显示该控件之前设置延迟。这可阻止当更新过快时，浏览器中发生控件闪烁。在默认情况下，延迟设置为 500ms(0.5s)，这意味着如果更新的时间少于 0.5s，则将不会显示 UpdateProgress 控件。在开发环境中，可以向应用程序添加人为延迟，从而确保 UpdateProgress 控件按预期方式工作。在 Page_Load 事件处理程序中添加以下代码，人为地创建 3s 的延迟：System.Threading.Thread.Sleep(2000)；出于本任务演示的需要，Page_Load 事件的处理程序有意引入了延迟。这是一个可选步骤，并且仅用于测试应用程序。在实际应用中，不用引入延迟。相反，延迟是由服务器通信量或需要花很长时间处理的服务器代码造成的，例如长时间运行的数据库查询。保存修改，然后按 Ctrl+F5 组合键在浏览器中查看页面。现在，由于向新页数据中移入了一个 3s 延迟，因此将会看到 UpdateProgress 控件。

当单击排序或分页按钮时，优化后的新闻搜索页运行效果如图 10-2 所示。

优化后的新闻搜索页的关键代码如下所示。

NewsPage.Master 的关键代码如下：

```
<%@ Master Language = "C#" AutoEventWireup = "true" CodeBehind = "NewsPage.master.cs"
Inherits = "Web.NewsPage" %>
…
<body>
    <form id = "form1" runat = "server">
    <asp:ScriptManager ID = "ScriptManager1" runat = "server">
```



图 10-2　优化后的新闻搜索页运行效果

```
    </asp:ScriptManager>
    <div id="container">
        ...
    </div>
    ...
    </form>
</body>
```

NewsSearch.aspx 的关键代码如下：

```
<%@ Page Language="C#" MasterPageFile="~/NewsPage.Master" AutoEventWireup="true"
CodeBehind="NewsSearch.aspx.cs" Inherits="Web.NewsSearch" Title="新闻搜索" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" runat="server">
    <div id="col">
        <div class="box">
            <div class="box_title">新闻搜索</div>
            <div class="box_text">
                <asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
                    <ContentTemplate>
                        <div>
                            排序方式:
                            ...
                        </div>
                        <div>
                            <ul id="title_ul">
                                ...
                            </ul>
                             <asp:Repeater ID="rpNews" runat="server" EnableViewState=
"false">
                            ...
                            </asp:Repeater>
                        </div>
                        <div id="page">
                            ...
                        </div>
                    </ContentTemplate>
                </asp:UpdatePanel>
```

```
                <asp: UpdateProgress  ID  =  " UpdateProgress1 "  runat  =  "  server "
AssociatedUpdatePanelID = "UpdatePanel1">
                    <ProgressTemplate>
                        正在更新……<br />
                        <asp:Image ID = " Image1" runat = " server" ImageUrl = " ~/App_
Themes/Default/Images/ajax_special_updater.gif" />
                    </ProgressTemplate>
                </asp:UpdateProgress>
            </div>
        </div>
    </div>
</asp:Content>
```

10.1.5 小结

（1）使用 ASP.NET AJAX 功能，可以生成丰富的 Web 应用程序，改进用户体验并提高 Web 应用程序的效率。

（2）ASP.NET AJAX 具有类似传统的 ASP.NET Web 应用程序的开发模式，使用起来非常方便。只要在 Visual Studio 中轻松拖放相应控件即可实现强大的客户端 AJAX 功能。

（3） ASP. NET AJAX 的 UpdatePanel、UpdateProgress 和 Timer 控件需要 ScriptManager 控件支持才能实现部分页更新。

（4）当对 UpdatePanel 进行属性设置时，不允许将 ChildrenAsTriggers 设置为 false 的同时将 UpdateMode 设置为 Always，否则会引发系统异常。

10.1.6 思考与练习

给新闻速览页 NewsList. aspx 增加局部刷新显示功能。